# Table of Contents

## I The Commercial Solvers

1. Basic Solver Usage .......................................................... 7
2. AlphaECP ............................................................................. 13
3. ANTIGONE ........................................................................... 27
4. BARON ............................................................................... 41
5. CONOPT ............................................................................ 57
6. CPLEX 12 ............................................................................ 109
7. DECIS ................................................................................ 161
8. DICOPT .............................................................................. 189
9. GloMIQO ............................................................................ 209
10. Gurobi 6.0 ......................................................................... 217
11. KNITRO ............................................................................. 247
12. LGO .................................................................................. 269
13. LINDO/LINDOGlobal ....................................................... 275
14. LocalSolver ....................................................................... 321
15. MINOS ............................................................................... 323
16. MOSEK .............................................................................. 355
17. MSNLP .............................................................................. 413
18. OQNLP .............................................................................. 427
19. PATH 4.7 ........................................................................... 441
20. PATHNLP ........................................................................... 475
21. SBB ................................................................................... 479
22. SNOPT .............................................................................. 489
23. SULUM 2.0 ......................................................................... 523
24. XA ................................................................................... 549
25. XPRESS ............................................................................. 559

## II The Free Solvers

26. BDMLP .............................................................................. 571
27. BENCH ............................................................................. 573
28. BONMIN and BONMINH ................................................... 583
29. CBC .................................................................................. 631
30. CONVERT .......................................................................... 649
31. Couenne ............................................................................ 655
32. Stochastic Programming (SP) with EMP ............................. 719
33. EXAMINER ......................................................................... 749
34. GAMS/AMPL .................................................................... 755
35. GAMSCHK ......................................................................... 757
36. GAMS/KESTREL - Remote Solver Execution on NEOS Servers ........................................................................ 777
37. GAMS/LINGO ..................................................................... 779
38. Gather-Update-Solve-Scatter (GUSS) ................................. 781
39. IPOPT and IPOPTH ........................................................... 787
40. JAMS - an EMP solver ....................................................... 831
41. Least Squares (LS) ............................................................. 849
42. LogMIP ............................................................................. 851
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>MILES</td>
<td>853</td>
</tr>
<tr>
<td>44</td>
<td>NLPEC</td>
<td>873</td>
</tr>
<tr>
<td>45</td>
<td>Optimization Services (OS)</td>
<td>887</td>
</tr>
<tr>
<td>46</td>
<td>OSICPLEX, OSUGROBI, OSIMOSEK, OSISOPLEX, OSIXPRESS</td>
<td>889</td>
</tr>
<tr>
<td>47</td>
<td>SCIP</td>
<td>891</td>
</tr>
</tbody>
</table>
Part I

The Commercial Solvers
Basic Solver Usage

Contents

1 Introduction ......................................................... 7
2 GAMS Options ...................................................... 7
3 The Solver Option File ............................................. 9
4 GAMS Dot Options .................................................. 10

1 Introduction

For the novice GAMS user, solver usage can be very simple: you run the model and inspect the listing file to see what the solution is. No knowledge of solver options or solver return codes is required. While this is enough for some users, most will quickly find they need some basic knowledge of how to control the solver and interpret the results. This section describes the GAMS options used to control a solver, how the GAMS solvers interpret these options, and how to interpret the model and solver status codes the solvers return.

While most solvers allow the user to set additional, solver-specific options, we will not be concerned with those here. In most cases, it is not necessary to use any solver-specific options: use of the generic GAMS options is sufficient. This carries an important benefit: since the solvers interpret the GAMS options in a consistent way, a GAMS option setting applies to all solvers, not just to a specific one.

2 GAMS Options

Options exist in two forms: global or model-specific. The option statement sets a global GAMS option, e.g.

```
option iterlim = 100;
```

while the model suffix sets a GAMS option for an individual model:

```
mymodel.iterlim = 10;
```

In addition, the default value of a global GAMS option can be set on the GAMS command line:

```
gams transport iterlim = 100
```

If a model-specific option is set, this takes precedence over the global setting. You can unset any model-specific option by assigning it the default value of `NA`:

```
mymodel.iterlim = NA;
```

The GAMS options for controlling solvers are described below. Included with each option is a description of how the option is interpreted by a GAMS solver.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bratio</td>
<td>GAMS uses the bratio value to determine if an advanced basis exists (see the GAMS User’s Guide). The result of this test is passed as a logical flag to the solver. Pivotal algorithms in GAMS solvers can make use of this advanced basis to speed up problem solution.</td>
</tr>
<tr>
<td>cheat</td>
<td>Cheat value for a global solver: Each new feasible solution must be at least cheat better than the previous one. This can speed up the search, but the search may miss the optimal solution. The cheat option is specified in absolute terms (like the optca option), so that non-negative values are appropriate for both minimization and maximization models. Using the cheat option invalidates any reporting of the dual bound or optimality gaps.</td>
</tr>
<tr>
<td>cutoff</td>
<td>Cutoff value: During a branch-and-bound search, the parts of the tree with an objective worse than cutoff are deleted. This can sometimes speed up the initial phase of the branch-and-bound algorithm, at the cost of ignoring feasible solutions whose value is worse than cutoff.</td>
</tr>
<tr>
<td>domlim</td>
<td>Sets the domain violation limit. Domain errors are evaluation errors in the nonlinear functions (e.g. $\sqrt{x}$ for $x &lt; 0$). When a domain violation occurs the domain error count is incremented; a solver will terminate if this count exceeds domlim and return solver status 5 EVALUATION ERROR LIMIT. Note that some solvers operate in a mode where trial function evaluations are performed; these solvers will not move to points at which evaluation errors occur, so the evaluation errors at trial points are not counted against the limit.</td>
</tr>
<tr>
<td>iterlim</td>
<td>Sets a limit on the number of iterations performed by the solver. If this limit is hit, the solver will terminate and return solver status 2 ITERATION INTERRUPT. It is up to the solver link to decide what it thought of as an “iteration”. For LP solvers, iterlim often refers to the number of simplex iterations (i.e., pivots). For a MIP solver, iterlim often refers to the cumulative number of simplex iterations over all solves of LP relaxations. For iterations that iterlim does not apply to (e.g., barrier iterations, major iterations in a nonlinear solver), solver specific options need to be set.</td>
</tr>
<tr>
<td>nodlim</td>
<td>Sets the branch-and-bound node limit. This is a limit on the total number of nodes in the tree, not on the number of active nodes. If this limit is hit, the solver will terminate and return solver status 4 TERMINATED BY SOLVER.</td>
</tr>
<tr>
<td>optca</td>
<td>Absolute optimality criterion for a global solver. The absolute gap is defined to be $</td>
</tr>
<tr>
<td>optcr</td>
<td>Relative optimality criterion for a global solver. The solver will stop as soon as it has found a feasible solution proven to be within optcr of optimal. The precise definition of optcr depends on the solver. GAMS and some solvers use the formula $</td>
</tr>
<tr>
<td>optfile</td>
<td>If nonzero, the solver should read an option file. If optfile=1 the name of the option file is solvername.opt. If optfile is between 2 and 999, the value determines the extension used. For example, optfile=2 implies solvername.op2, optfile=67 implies solvername.67, optfile=525 implies solvername.525, etc.</td>
</tr>
<tr>
<td>prioropt</td>
<td>Instructs the solver to use the priority branching information passed by GAMS through variable suffix values variable.prior. If and how priorities are used is solver-dependent.</td>
</tr>
<tr>
<td>reslim</td>
<td>Sets the time limit in seconds. If this limit is hit, the solver will terminate and return solver status 3 RESOURCE INTERRUPT. The solver should start the clock fairly early, so that time required to read in the problem and do any reformulation, preprocessing, or presolving is included in the time limit. For a multi-threaded solve, the time limit applies to the wall clock time.</td>
</tr>
<tr>
<td>sysout</td>
<td>If sysout=on is set, GAMS will echo all the solver messages to the GAMS listing file. This is useful for debugging or to get additional information about a solver run. Normally, only those messages flagged by the solver as destined for the listing file get listed. sysout exists only as a global option, and can be set from the command line using an integer (e.g., sysout=1)</td>
</tr>
</tbody>
</table>
This option controls the number of threads to be used by a solver. When set to 0, the number of threads is set to the number of CPU cores. If set to a negative number, it specifies the number of cores that should not be used.

**tryint**
Signals the solver to make use of a partial or near-integer-feasible solution stored in current variable values to get a quick integer-feasible point. If or how tryint is used is solver-dependent.

**workfactor**
Specifies a factor to be applied to the solver-computed memory estimate. E.g., setting workfactor=2 doubles the memory estimate. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect. In cases where workfactor and workspace are both specified, the workspace setting takes precedence.

**workspace**
Specifies the amount (in MB) of memory the solver should allocate. This is used to override the solver-computed memory estimate. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect, or may be used as a memory limit. workspace exists only as a model-specific option.

Refer to Section E.3 of the GAMS User’s Guide for information on default values for these options.

## 3 The Solver Option File

To specify solver-specific options, it is necessary to use a solver option file. Two things are required to do this: you must create an option file having a proper name, and you must tell the solver to read and use this option file.

To tell a solver to use an option file, you can set the optfile model suffix to a positive value. For example,

```gams
model mymodel /all/
mymodel.optfile = 1;
solve mymodel using nlp maximizing dollars;
```

The option file takes its name from the solver being used: solvename.XXX, where solvename is the name of the solver that is specified, and the suffix XXX depends on the value to which the model suffix optfile has been set. If its value is 1, the suffix is opt. For example, the option file for CONOPT is called conopt.opt; for DICOPT, it is dicopt.opt.

If you do not set the .optfile suffix to a nonzero value, no option file will be used even if one exists.

To allow different option file names for the same solver, the .optfile model suffix can take on values between 2 and 999. In this case, the option file extension is computed from the .optfile value by replacing the characters in opt with the digits in the characters in the .optfile value, starting from the right. For example,

<table>
<thead>
<tr>
<th>optfile model suffix value</th>
<th>Name of option file</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No option file used</td>
</tr>
<tr>
<td>1</td>
<td>solvename.opt</td>
</tr>
<tr>
<td>2</td>
<td>solvename.op2</td>
</tr>
<tr>
<td>3</td>
<td>solvename.op3</td>
</tr>
<tr>
<td>10</td>
<td>solvename.o10</td>
</tr>
<tr>
<td>91</td>
<td>solvename.o91</td>
</tr>
<tr>
<td>100</td>
<td>solvename.100</td>
</tr>
<tr>
<td>999</td>
<td>solvename.999</td>
</tr>
</tbody>
</table>

For example, setting mymodel.optfile to 23 will result in the option file conopt.o23 being used for CONOPT, and dicopt.o23 being used for DICOPT.

The format of the options file is not completely standard and changes marginally from solver to solver. This section illustrates some of the common features of the option file format. Please check the solver-specific documentation before using an option
Blank lines in an option file are ignored. Each nonblank line falls into one of two categories:

- a comment line
- an option specification line

A comment line begins with an asterisk (*) in the first column, is not interpreted by either GAMS or the solver, and is used purely for documentation. Each option specification line can contain only one option. The format for specifying options is as follows:

```
keyword(s) [modifier] [value]
```

The keyword may consist of one or more words and is not case sensitive. The value might be an integer, a real, or a string. All solvers will accept real numbers expressed in scientific (i.e. E) format. Note that not all options require modifiers or values.

Any errors in the spelling of keyword(s) or modifiers will lead to that option being misunderstood and therefore ignored. Errors in the value of an option can result in unpredictable behavior. When detected, errors are either ignored or pushed to a default or limiting value, but not all can or will be detected. Option values should be chosen thoughtfully and with some care.

Consider the following CPLEX options file,

```
* CPLEX options file
barrier
crossover 2
```

The first line begins with an asterisk and therefore contains comments. The first option specifies the use of the barrier algorithm to solve the linear programming problem, while the second option specifies that the crossover option 2 is to be used. Details of these options can be found in the CPLEX section of this manual.

Consider the following MINOS options file,

```
* MINOS options file
scale option 2
completion partial
```

The first option sets the scale option to a value of 2. In this case, the keyword 'scale option' consists of two words. In the second line, the completion option is set to partial. Details of these options can be found in the MINOS section of this manual.

### 4 GAMS Dot Options

Dot options in a solver option file allow users to associate values to variables and equations using the GAMS name of the variables and equations. The general syntax of a dot option in the option file is as follows:

```
(variable/equation name).optionname (value)
```

Dot options can be specified for all, a block, a slice, and a single variable and equation. Please note that a specific dot option may only apply to variables or equations (e.g. the GAMS/Gurobi dot option prior applies to variables only). The following example makes the use of the dot option clear.

For example, suppose we have a GAMS declaration:

```gams
Set i /i1*i5/;
Set j /j2*j4/;
Variable v(i,j);
Equation e(i,j);
```

Consider the following lines in an option file with the imaginary option name dotopt:
### Line in option file

<table>
<thead>
<tr>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables.dotopt 1</td>
</tr>
<tr>
<td>Sets the value of all variables to 1</td>
</tr>
<tr>
<td>equations.dotopt 2</td>
</tr>
<tr>
<td>Sets the value of all equations to 2</td>
</tr>
<tr>
<td>v.dotopt 3</td>
</tr>
<tr>
<td>Sets the value of the variables in block v to 3</td>
</tr>
<tr>
<td>e.dotopt(∗,∗) 4</td>
</tr>
<tr>
<td>Sets the value of the equations in block e to 4</td>
</tr>
<tr>
<td>v.dotopt(∗,′j2′) 5</td>
</tr>
<tr>
<td>Sets the value of the variables v that have j2 in the second index position (slice) to 5</td>
</tr>
<tr>
<td>e.dotopt(′i3′,∗) 6</td>
</tr>
<tr>
<td>Sets the value of the equations e that have i3 in the first index position (slice) to 6</td>
</tr>
<tr>
<td>w.dotopt(′i2′) 7</td>
</tr>
<tr>
<td>Sets the value of the single variables v(′i2′) to 7</td>
</tr>
<tr>
<td>e.dotopt(′i3′,′j3′) 8</td>
</tr>
<tr>
<td>Sets the value of the single equations e(′i3′,′i3′) to 8</td>
</tr>
</tbody>
</table>

The values of the dot option are applied in correspondence to the sequence in which they appear in the option file. In the current example, the values of dotopt for the equation e would be as follows:

<table>
<thead>
<tr>
<th>e.dotopt</th>
<th>i1</th>
<th>i2</th>
<th>i3</th>
</tr>
</thead>
<tbody>
<tr>
<td>j2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>j3</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>j4</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
1 Introduction

AlphaECP is a Mixed-Integer Non-Linear Programming (MINLP) solver based on the extended cutting plane (ECP) method. The solver can be applied to general MINLP problems and it can ensure global optimal solutions for pseudo-convex MINLP problems.

The ECP method is an extension of Kelley’s cutting plane method, which was originally given for convex NLP problems [1]. The method only requires the solution of a MIP sub-problem in each iteration. The MIP sub-problems can be solved to optimality, to feasibility, or only to an integer relaxed solution in intermediate iterations. This makes the ECP algorithm efficient and easy to implement. Further information about the underlying algorithm can be found in [5] and [2–4, 6].

Further developments of the GAMS/AlphaECP algorithm have introduced additional functionality. A NLP solver can now be called at MIP solutions. This improves AlphaECP’s ability to find feasible and accurate solutions, especially for MINLP problems which mainly contain continuous variables. Furthermore, a heuristic that reselects cutting planes during the iteration procedure can be used to improve the capability of solving non-convex problems.

1.1 Licensing and software requirements

Users need to have a GAMS/AlphaECP license in order to use GAMS/AlphaECP. In addition, a licensed MIP solver is required for solving the mixed integer sub-problem, and a licensed NLP solver is required if the NLP option is used.
1.2 Running AlphaECP

AlphaECP solves MINLP models. If AlphaECP is not specified as the default solver for these models it can be invoked by issuing the following command before the solve statement:

```
option minlp=alphaecp, miqcp=alphaecp;
```

In principle AlphaECP can also handle NLP models, but it is more suitable for MINLP problems. However, when combined with an NLP solver it can find solutions the NLP solver cannot find by itself. In this case it acts as a good starting point generator. If you want to solve NLPs with AlphaECP you need to trick the GAMS system by solving your NLP as an MINLP:

```
solve mynlpmodel minimizing obj using minlp;
```

Constraint violations are reported throughout the progress of AlphaECP and at the end of the algorithm. The violation is reported for the non-linear constraints only. The violation of the linear constraints is subject to the feasibility tolerance of the MIP/NLP solver.

2 GAMS/AlphaECP Output

The log output below is obtained for the MINLP model fuel.gms from the GAMS model library:

```
Welcome to Alpha-ECP v2.10.06
MINLP Problem Solver using the Extended Cutting Plane Approach.
Method development - T.Westerlund, Abo Akademi University, FIN
Algorithm implementation - T.Lastusilta, Abo Akademi University, FIN

Minimization problem: "fuel.gms"
The GAMS-model has in total 39 elements of which 15% are non-linear(NL)
included in 16 constraints of which 25% are NL
The NL constraint signs: =E=(3), =G=(1), =L=(0)
The number of variables in NL elements are 6 from a total of 16
variables: Continuous(13), Binary(3), Integer(0)

Using following settings
AlphaECP option file optfile=0
Time limit for AlphaECP (in seconds) reslim=1000
Solve link for NLP and MIP sub-solver solvelink=5
Solver trace file solvetrace=(Inactive)
Cutting plane strategy (0-3) CUTdelcrit=3
Cut generation pace CUThrcuts=0
Updating multiplier if MIP is infeasible ECPbeta=1.3
Write encountered solutions to gdx files ECPdumpsol=0
Updating multiplier when verifying solution ECPgamma=2
Maximum number of AlphaECP iterations ECPiterlim=-1
Level of AlphaECP output to statusfile (0-4) ECPloglevel=0
Master strategy (0=User 1=Convex) ECPmaster=0
User specified start-point (0-3) ECPstart=3
Return solution (1.MIP/2.NLP/...) ECPretsol=2
AlphaECP strategy (1-5) ECPstrategy=2
Upper limit of considered MIP solutions per MIP call MIPnrsols=50
Relative MIP gap in intermediate sub-problems (0->1.0) MIPoptcr=1.00
Initial MIPoptcr interval before MIPoptcr reduction MIPoptcrlim=200
```
Strategy for multiple MIP solutions: MIPsolstrat=1
MIP solver for sub-problems and . option file number: MIPsolver=cplex.0
NLP strategy. Inactive:0 Active strategy:1-5: NLPcall=5
NLP solver call at next (incremental) iteration: NLPcalliter=0
NLP time limit per call (in seconds or auto=0): NLPreslim=30
NLP solver for sub-problems and . option file number: NLPsolver=conopt.0
Constraint tolerance: TOLepsg=0.001
Distance tolerance for a new linearization: TOLepsz=0.1
Gradient tolerance: TOLgrad=1e-006
Infinity bound (MIP variable bound): TOLinfbnd=1e+010

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Stepcode</th>
<th>Number</th>
<th>Point</th>
<th>Alpha</th>
<th>OPT</th>
<th>Movement</th>
<th>Violation</th>
<th>Maximum</th>
<th>MIPobjval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1.8E+003</td>
<td>NA</td>
</tr>
<tr>
<td>1</td>
<td>SAFGI</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9.3E+003</td>
<td>0</td>
<td>-0</td>
<td>8566.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FOUND SOLUTION: 8566.12 (NLP) in 1 sec.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SAFH</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6.6E+003</td>
<td>4</td>
<td>1.8E+003</td>
<td>4844.02</td>
</tr>
<tr>
<td>3</td>
<td>SAFH</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>8.4E+003</td>
<td>3</td>
<td>1.8E+003</td>
<td>7031.72</td>
</tr>
<tr>
<td>4</td>
<td>SAFH</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1E+003</td>
<td>2</td>
<td>1.8E+003</td>
<td>10157</td>
</tr>
<tr>
<td>5</td>
<td>SAH</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7E+002</td>
<td>11925</td>
</tr>
<tr>
<td>6</td>
<td>SAH</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1.1E+003</td>
<td>2</td>
<td>3.5E+002</td>
<td>11925</td>
</tr>
<tr>
<td>7</td>
<td>SAH</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1.9E+003</td>
<td>2</td>
<td>5.3E+002</td>
<td>11925</td>
</tr>
<tr>
<td>8</td>
<td>SAH</td>
<td>9</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>8.8E+002</td>
<td>3</td>
<td>2.6E+002</td>
<td>11925</td>
</tr>
<tr>
<td>9</td>
<td>SAH</td>
<td>12</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>8.8E+002</td>
<td>3</td>
<td>1.3E+002</td>
<td>11925</td>
</tr>
<tr>
<td>10</td>
<td>SAH</td>
<td>15</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>4.4E+002</td>
<td>3</td>
<td>66</td>
<td>11925</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>AIJ</td>
<td>101</td>
<td>66</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00067</td>
<td>11925</td>
</tr>
<tr>
<td></td>
<td>Pointusage</td>
<td>44/66</td>
<td>Cutusage</td>
<td>111/172</td>
<td>(</td>
<td>0.82</td>
<td>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>AH</td>
<td>111</td>
<td>44</td>
<td>0</td>
<td>0</td>
<td>1.4E+003</td>
<td>3</td>
<td>76</td>
<td>9926.15</td>
</tr>
<tr>
<td>94</td>
<td>AI</td>
<td>113</td>
<td>45</td>
<td>7</td>
<td>0</td>
<td>1.4E+003</td>
<td>0</td>
<td>0.00067</td>
<td>11925</td>
</tr>
<tr>
<td>95</td>
<td>AH</td>
<td>113</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>5.3E+002</td>
<td>2</td>
<td>58</td>
<td>10898</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>AI</td>
<td>117</td>
<td>48</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00067</td>
<td>11925</td>
</tr>
<tr>
<td>105</td>
<td>AIJ</td>
<td>117</td>
<td>48</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00067</td>
<td>11925</td>
</tr>
</tbody>
</table>

AlphaECP: Iteration procedure terminated normally

Problem : fuel.gms
Solver Status : Normal Completion
Model Status : Locally Optimal
Exit comment : No Issues
Final solution : NLP
Objective value : 8566.1189616876654
Max constraint (1) : -0
Alternative solution : MIP
Alt. objective value : 8566.1189616876654
Max constraint (1) : -0
Time used (seconds) : 0.97
Time limit (seconds) : 1000
Iterations used : 105
Iteration limit : -1
Function evaluations : 484
Gradient evaluations : 185
Domain violations : 0
Gradients unusable : 0
<table>
<thead>
<tr>
<th>AlphaECP bound violations</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECP time usage</td>
<td>3.2 %</td>
</tr>
<tr>
<td>NLP time usage</td>
<td>3.2 %</td>
</tr>
<tr>
<td>MIP time usage</td>
<td>93.6 %</td>
</tr>
<tr>
<td>Optimal/total MIPs</td>
<td>24/105</td>
</tr>
<tr>
<td>NLP solver calls</td>
<td>8</td>
</tr>
</tbody>
</table>

In every iteration, information about the MIP problem and modifications to it is given in 10 columns. Here is a description of the different columns:

**Iteration**: Iteration identifier.

**Stepcode, Problems**: Letter for what actions were taken in this iteration, i.e. MIP problem modifications before the next iteration.

- **A**: MIP solver feasible.
- **B**: MIP solver feasible after moving cutting planes, i.e. alpha update.
- **C**: MIP solver feasible after moving cutting planes close to their generation point. The movement is done to make it easier to satisfy nonlinear equality constraints.
- **D**: Line search was successful (in ECPstrategy 3).
- **E**: Line search failed (in ECPstrategy 3).
- **F**: A NLP solver was called.
- **G**: Found a MINLP solution.
- **H**: Added linearization(s) to the next MIP problem.
- **I**: Updated alpha values and possibly added linearizations.
- **J**: All cutting planes are valid underestimators for the pseudo-convex constraints, except for the nonlinear objective function constraint.
- **K**: The nonlinear objective function constraint value and MIP solution value differ more than $\varepsilon_f$. A linearization was done to reduce the difference (in ECPstrategy 3).
- **L**: Removed all temporal linearizations.
- **M**: Domain violation(s), some of the constraint could not be evaluated.
- **N**: Some cutting plane(s) could not be generated because of gradient problems.
- **O**: No cutting planes could be generated.
- **P**: Reselecting cuts because cutting planes are repeatedly moved close to their generation point.
- **Q**: Added temporal linearization(s).
- **R**: Failed to add temporal linearization(s).
- **S**: MIP solver strategy to find encountered solutions selected.
- **T**: MIP solver strategy to require MIPnrsols solutions selected.
- **U**: MIP solver strategy to require MIPnrsols solutions with a MIPoptcr $\leq 0.2$ selected.

**Number of Cuts**: The number of cutting planes the solved MIP problem had.

**Point usage**: Number of points used to generate the cuts in the solved MIP problem.

**Alpha Upd.**: The number of times the alpha values has been increased.

**OPTCR**: Requirement of the relative distance to the relaxed MIP solution for the current MIP solution.

**Movement Norm**: The Euclidean norm of the current and previous MIP solution.

**Viol Cons**: Number of unsatisfied (violating) nonlinear constraints.

**Maximum Violation**: The most violating nonlinear constraint value.
MIPobjval/NLobjval: MIP or nonlinear objective variable value depending on ECPstrategy setting.

The cut reselection heuristic is called in the following cases:

1. If the MIP solver would otherwise return infeasible.
2. When the violation is not reducing, but the cutting planes are repeatedly moved close to their generation point.
3. When the violation is not reducing and domain violations are repeatedly encountered.

The heuristic reselects cutting planes in different ways, but always ensures that the same point cannot be found twice.

| Point usage 6/90 | Cut usage 15/341 (0,135) |

Point usage informs how many points of all usable points have been used to generate the cutting planes. Cut usage tells how many cuts of all usable cuts have been used. The first number in (0,135) tells how many cuts is required by the user, see CUTnrcuts and the second number gives the sum of added and removed cuts, i.e. a measure of how much the MIP problem has been modified. AlphaECP may fix some cuts and remove points and cuts during the cut reselection procedure in order to save memory.

At the end of each solve AlphaECP gives a summary which includes Problem, Solver Status, Model Status, etc. Note the following lines:

- **Exit comment** may give further information than solverstatus on why the solution procedure stopped.
- **Domain violations** (function evaluation failed) or **Gradients unusable** (all gradients < TOLgrad) might be caused by poor variable bounds.
- **Alphamax bound violations** inform the user how many times an alphamax value was calculated to be > 10^{154} and was reset to 10^{154}.

### 3 Notes about Options

To instruct AlphaECP to read an option file you may use `ModelName.OptFile = 1`. The name of the option file is in this case `alphaecp.opt`. For further information, see Section **The Solver Option File** in Basic Solver Usage. AlphaECP supports the GAMS parameter `reslim`, however, other GAMS parameters are passed on to the sub-solvers. Furthermore, you may also pass additional sub-solver specific options to the sub-solvers. For example, if you want to use all available threads and sub-solver CPLEX in opportunistic parallel search mode, then you may specify this in a GAMS model, in a similar way, as follows:

```plaintext
Model m / all /;
Option threads=0;
m.optfile=1;
$echo MIPsolver cplex.1 > alphaecp.opt
$echo parallelmode -1 > cplex.opt
Solve m using MINLP minimizing objvar;
```

The following information is worth noting when you are interested in AlphaECP options. A linearization of a nonlinear constraint is called a cutting plane or cut. Here a point refers to the variable levels. Global optimality can be guaranteed for pseudo-convex problems. However, if the objective variable is in a nonlinear constraint and pseudo-convex, then ECPstrategy \( \geq 3 \) needs to be used to guarantee global optimality (because one non-linear equality constraint makes a problem non-pseudo-convex, and hence also non-convex). The basic options might significantly impact the solution procedure, and the best values are likely to be problem specific. The user is therefore encouraged to try different values for the basic options.
4 Summary of AlphaECP Options

4.1 Basic options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUTnrcuts</td>
<td>Cut generation pace</td>
<td>0</td>
</tr>
<tr>
<td>ECPmaster</td>
<td>Master strategy (0=User 1=Convex)</td>
<td>0</td>
</tr>
<tr>
<td>MIPnrsols</td>
<td>Upper limit of considered MIP solutions per MIP call</td>
<td>50</td>
</tr>
<tr>
<td>MIPsolstrat</td>
<td>MIP solution collection strategy</td>
<td>1</td>
</tr>
<tr>
<td>MIPsolver</td>
<td>MIP solver for sub-problems and . option file number</td>
<td>GAMS MIP solver</td>
</tr>
<tr>
<td>NLPsolver</td>
<td>NLP solver for sub-problems and . option file number</td>
<td>GAMS NLP solver</td>
</tr>
<tr>
<td>reslim</td>
<td>Time limit for AlphaECP (in seconds)</td>
<td>GAMS reslim</td>
</tr>
</tbody>
</table>

### 4.2 Algorithmic options for advanced users

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUTdelcrit</td>
<td>Cutting plane strategy</td>
<td>3</td>
</tr>
<tr>
<td>ECPbeta</td>
<td>Updating multiplier if MIP is infeasible</td>
<td>1.3</td>
</tr>
<tr>
<td>ECPdumpsol</td>
<td>Write encountered solutions to gdx files</td>
<td>0</td>
</tr>
<tr>
<td>ECPgamma</td>
<td>Updating multiplier when verifying solution</td>
<td>2.0</td>
</tr>
<tr>
<td>ECPiterlim</td>
<td>Maximum number of AlphaECP iterations</td>
<td>-1</td>
</tr>
<tr>
<td>ECPloglevel</td>
<td>Level of AlphaECP output to statusfile</td>
<td>0</td>
</tr>
<tr>
<td>ECPpcostratgy</td>
<td>Pseudo-convex objective function strategy</td>
<td>3</td>
</tr>
<tr>
<td>ECPretsol</td>
<td>Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE)</td>
<td>2</td>
</tr>
<tr>
<td>ECPstart</td>
<td>User specified start-point</td>
<td>3</td>
</tr>
<tr>
<td>ECPstrategy</td>
<td>AlphaECP strategy</td>
<td>2</td>
</tr>
<tr>
<td>solvelink</td>
<td>Solvelink for NLP and MIP sub-solver</td>
<td>5</td>
</tr>
<tr>
<td>solvetrace</td>
<td>Filename of solvetrace file</td>
<td></td>
</tr>
<tr>
<td>solvetracetime</td>
<td>Time interval when a trace record is written</td>
<td>1</td>
</tr>
<tr>
<td>TOLepsf</td>
<td>Pseudo-convex objective function termination tolerance</td>
<td>1e-3</td>
</tr>
<tr>
<td>TOLepsg</td>
<td>Constraint tolerance</td>
<td>1e-3</td>
</tr>
<tr>
<td>TOLepsz</td>
<td>Distance tolerance for a new linearization</td>
<td>1e-1</td>
</tr>
<tr>
<td>TOLgrad</td>
<td>Gradient tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>TOLinfbnd</td>
<td>Infinity bound (MIP variable bound)</td>
<td>1e10</td>
</tr>
</tbody>
</table>

### 4.3 MIP Solver related options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPloglevel</td>
<td>Level of MIP solver output</td>
<td>0</td>
</tr>
<tr>
<td>MIPoptcr</td>
<td>Relative MIP gap in intermediate sub-problems</td>
<td>1.0</td>
</tr>
<tr>
<td>MIPoptcrlim</td>
<td>Initial MIPoptcr interval before MIPoptcr reduction</td>
<td>200</td>
</tr>
</tbody>
</table>

### 4.4 NLP Solver related options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLPcall</td>
<td>NLP strategy</td>
<td>5</td>
</tr>
<tr>
<td>NLPcalliter</td>
<td>NLP solver call at next (incremental) iteration</td>
<td>0</td>
</tr>
</tbody>
</table>
5 Detailed Descriptions of AlphaECP Options

CUTdelcrit (integer): Cutting plane strategy
   (default = 3)
   0 Do not remove any valid cuts.
   1 As 0 and allow temporary cuts at semirandom points if normal cuts can not be made.
   2 Allow temporary cuts and cut reselection, and use memory to save points and cuts.
   3 As 2 and call the reselection heuristic before termination to improve the solution.

CUTnrcuts (real): Cut generation pace
   The number of linearizations that are generated during an iteration can be chosen by AlphaECP, proportional to
   the number of violating constraints, or can be determined by a fixed amount. Furthermore, the cut reselection
   CUTdelcrit >=2 adds cuts to the problem so that the requested cut generation pace is taken into consideration.
   (default = 0)
   0 Let AlphaECP decide.
   0<n<1 Number of linearizations = n* the number of linearizations that is possible to generate.
   >1 Specifies the number of linearizations to generate.

ECPbeta (real): Updating multiplier if MIP is infeasible
   In case of an infeasible MIP solution, the invalid cuts are updated with the ECPbeta multiplier.
   (default = 1.3)

ECPdumpsol (integer): Write encountered solutions to gdx files
   (default = 0)
   0 No.
   1 Solutions that the NLP solver found.
   2 Solutions that the NLP or MIP solver found.

ECPgamma (real): Updating multiplier when verifying solution
   If a MINLP solution is obtained but some cuts are not valid underestimators they are updated with the ECPgamma
   multiplier in order to make them into valid underestimators.
   (default = 2.0)

ECPiterlim (integer): Maximum number of AlphaECP iterations
   This is the maximum number of iterations given to AlphaECP to perform the optimization. Value -1 deactivates
   the AlphaECP iteration limit.
   (default = -1)
   ~1 No limit.
   >=0 Specifies an iteration limit.

ECPloglevel (integer): Level of AlphaECP output to statusfile
   (default = 0)
   0 No additional output to statusfile.
1. Report solutions. Report all encountered solutions with their corresponding variable levels.
2. Report main actions at iteration level (available for minimization problems).
4. Full reporting. Report the main actions taken, the linearizations, function values, and solution points for every iteration and line search details (available for minimization problems).

**ECPmaster (integer):** Master strategy (0=User 1=Convex)

The master strategy sets some options in order to solve a model with specific characteristics more efficiently. The affected options are noted in the log output. The set options takes precedence over the value set by the user for the affected options.

(default = 0)

0. Use only user defined options.
1. The model is convex. Set option ECPstrategy and CUTdelcrit.

**ECPpcostrategy (integer):** Pseudo-convex objective function strategy

(default = 3)

1. Remove support. Remove old support planes when a new pseudo-convex problem is formed.
2. Replace support. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed.
3. Remove support and line search. Remove old support planes when a new pseudo-convex problem is formed and perform a line search when it is possible.
4. Replace support and line search. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed and perform a line search when it is possible.

**ECPretsol (integer):** Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE)

The reported solution can be extracted from either the MIP or NLP solver result. If the MIP solution is returned only the primal values are available.

(default = 2)

1. Choose MIP solution if it is available.
2. Choose NLP solution if it is available.
3. Choose the solution with the best tolerance.
4. Choose the solution with the best objective value.

**ECPstart (integer):** User specified start-point

Define which variable levels are used when the optimization is started.

(default = 3)

0. Do not use a start-point; start the algorithm by solving the linear part (MIP) of the problem.
1. Use the user specified startpoint, but adjust the variable levels with a small value.
2. Use the exact start-point set by the user.
3. Use the exact start-point if linearly feasible; else adjust variable levels with a small value.

**ECPstrategy (integer):** AlphaECP strategy

(default = 2)

1. Convex strategy. Ensures global optimality for problems with convex objective function and convex constraints.
3 Pseudo-convex objective. Ensures global optimality for problems with pseudo-convex objective function and pseudo-convex constraints. The reformulation of a non-linear objective function into a constraint must be done in a specific way. The requirement is that the objective variable must be in a linear part of the non-linear function. Assuming that the minimized or maximized variable is called objvar, the reformulation can be done as follows: (objective function expression) - objvar =E= 0.

Furthermore, this strategy can effectively use a feasible start-point.

4 Pseudo-convex objective, but first complete with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)

5 Pseudo-convex objective, but find the first solution with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)

MIPloglevel (integer): Level of MIP solver output

By default the detailed log of the MIP solver is suppressed in the AlphaECP log stream. If this option is turned on and the GAMS LogOption is set to 1 or 3, the MIP log will be merged into the AlphaECP log.

(default = 0)

0 No output.

1 MIP solver log goes to GAMS log.

MIPnrsols (integer): Upper limit of considered MIP solutions per MIP call

When the MIP solver returns several solutions the most suitable solution is chosen. Many times the solutions from the MIP solver are similar and a larger number might help to find a feasible MINLP solution if the constraints are almost satisfied. See MIPsolstrat to change the solution collection strategy.

(default = 50)

MIPoptcr (real): Relative MIP gap in intermediate sub-problems

The relative stopping tolerance sent to the MIP solver for intermediate MIP problems. Note that the MIPoptcr value is decreased automatically to zero during the optimization.

(default = 1.0)

MIPoptcrlim (integer): Initial MIPoptcr interval before MIPoptcr reduction

The MIPoptcr parameter is reduced in steps: From 1 to 0.5 to 0.3 to 0.2 to 0.1 to 0.0. The first reduction is at iteration MIPoptcrlim. MIPoptcrlim defines a step reduction at specific iterations (next reduction at iteration = the iteration number for this reduction multiplied by two). Note that a step reduction can also be caused by other reasons. If MIPoptcrlim is 200 then MIPoptcr is reduced at the following iterations: 200, 400, 800, etc.

(default = 200)

MIPsolstrat (integer): MIP solution collection strategy

(default = 1)

0 Instruct MIP solver to return only one solution.

1 Instruct MIP solver to return any solutions encountered during MIP procedure.

2 Instruct MIP solver to search for solutions to obtain requested number MIPnrsols solutions.

3 As 2, but also require the solutions to fulfill MIPoptcr >= 0.2.

4 Let AlphaECP decide.

MIPsolver (string): MIP solver for sub-problems and . option file number

solver[n] Solver is the name of the GAMS MIP solver and n is the integer corresponding to optfile. The option file is appended to the option file, that is written by AlphaECP. Hence, the specified options take precedence over the options set by AlphaECP. If .n is missing, the optfile is treated as zero i.e. the MIP solver will not look for a options file specified by the user. This option can be used to overwrite the default that uses the MIP solver specified with an Option NLP = solver; statement or the default GAMS solver for NLP.

(default = GAMS MIP solver)
NLPcall (integer): NLP strategy

Determine when the NLP solver is called.

(default = 5)

0 No output.
1 Call the NLP solver at end of AlphaECP algorithm.
2 As 1 and when a better solution is found.
3 As 2 and when the same integer solution is encountered NLPlimsameint times.
4 Let AlphaECP decide.
5 Let AlphaECP decide and add noise to the variable levels before call.

NLPcalliter (integer): NLP solver call at next (incremental) iteration

Specify an iteration interval for the NLP solver calls.

(default = 0)

NLPlimsameint (integer): NLP call after a number of recurring integer solutions

If the same integer solution is encountered NLPlimsameint times in a row then the NLP solver is called. The counter is reset after the NLP solver is called.

(default = 5)

NLPloglevel (integer): Level of NLP solver output

By default the detailed log of the NLP solver is suppressed in the AlphaECP log stream. If this option is turned on and the GAMS LogOption is set to 1 or 3, the NLP log will be merged into the AlphaECP log.

(default = 0)

0 No output.
1 NLP solver log goes to GAMS log.

NLPreslim (real): NLP time limit per call

The time limit (in seconds) given to the chosen NLP solver at each NLP solver call. Setting this option to 0 calculates a time limit which is relative to the problem size.

(default = 0)

NLPsolver (string): NLP solver for sub-problems and . option file number

Solver[.n] Solver is the name of the GAMS NLP solver that should be used in the root node, and n is the integer corresponding to optfile. If .n is missing, the optfile is treated as zero, i.e., the NLP solver will not look for an options file. This option can be used to overwrite the default that uses the NLP solver specified with an Option NLP = solver; statement or the default GAMS solver for NLP.

(default = GAMS NLP solver)

reslim (real): Time limit for AlphaECP (in seconds)

(default = GAMS reslim)

solvelink (integer): Solvelink for NLP and MIP sub-solver

(default = 5)

1 Call NLP and MIP solver via script.
2 Call NLP and MIP solver via module.
5 Call NLP and MIP solver in memory.

solvetrace (string): Filename of solvetrace file

solvetracetime (real): Time interval when a trace record is written
(default = 1)

**TOLepsf (real):** Pseudo-convex objective function termination tolerance

Maximum allowed absolute difference between the nonlinear and the MIP objective function value (used only in ECPstrategy 3).

(default = 1e-3)

**TOLepsg (real):** Constraint tolerance

The nonlinear constraint tolerance defines the maximum value that a nonlinear constraint may violate. For example, a constraint required to be zero may hold a value +/- TOLepsg at a solution.

(default = 1e-3)

**TOLepsz (real):** Distance tolerance for a new linearization

The maximum perpendicular distance between a valid cutting plane and its generation point (MIP solution).

(default = 1e-1)

**TOLgrad (real):** Gradient tolerance

The absolute value of a gradient’s partial derivative must be above TOLgrad value in order for it to be considered nonzero.

(default = 1e-6)

**TOLinfbnd (real):** Infinity bound (MIP variable bound)

All variables must have a positive and a negative finite bound in order to ensure a bounded MIP problem. The finite bound value TOLinfbnd will be applied to single or double unbounded variables.

(default = 1e10)

6 FAQ

- What are good settings to solve a convex problem?
  
  Use ECPmaster 1.

- What are good settings if the solution speed is essential?
  
  Try ECPstrategy 1 and CUTdelcrit 1 to see if using multiple threads for the MIP solver improves the solution speed. However, there is considerable chance that a feasible solution for a non-convex problem with nonlinear equality constraints cannot be found.

- What are good settings when the solution quality is essential?
  
  Use NLPcalliter 1 and MIPSolstrat 4 or 3, and also try different values, for CUTnrcuts option, for example 0.1.

- The objective function is non-linear, should the default ECPstrategy be used?
  
  If the objective function constraint can be written in the required form of ECPstrategy 3 then this strategy may find a better solution. If the constraints and the objective function are pseudo-convex the global optimal solution will be found.

Bibliography


1 Introduction

ANTIGONE, Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations, is a deterministic general mixed-integer nonlinear global optimization framework [27–31]. MINLP is defined:

\[
\begin{align*}
\text{min} & \quad f_0(x, y, z) \\
\text{s.t.} & \quad h_m^{LO} \leq f_m(x, y, z) \leq h_m^{UP} \quad \forall m \in \{1, \ldots, M\} \\
& \quad x \in \mathbb{R}^C; \ y \in \{0, 1\}^B; \ z \in \mathbb{Z}^I
\end{align*}
\]  

(MINLP)

where \( C, B, I, \) and \( M \) represent the number of continuous variables, binary variables, integer variables, and constraints, respectively. Parameters vectors \( h_m^{LO} \) and \( h_m^{UP} \) bound the constraints. We assume that it is possible to infer finite bounds
Given an MINLP optimization problem, ANTIGONE reformulates the model, detects special structure in the reformulated MINLP, solves the optimization problem, and returns the model with respect to the original problem variables \([x^L, x^U]\) on the variables participating in nonlinear terms \(f_m\) and that the image of \(f_m\) is finite on \(x\). Typical expressions for \(f_0(x, y, z)\) and \(f_m(x, y, z)\) are:

\[
f_m(x, y, z) = c_m + a_m^T [x; y; z] + [x; y; z]^T Q_m [x; y; z]
\]

\[
+ \sum_{s=1}^{S_m} c_{s,m} \cdot \prod_{c=1}^{C_{s,m}} x^{p_{s,m,c}} + \sum_{e=1}^{E_m} c_{e,m} \cdot e^x + \sum_{\ell=1}^{L_m} c_{\ell,m} \cdot \log x
\]

where the powers \(p_{s,m,c}\) are constant reals; \(c_m, a_m, Q_m, c_{s,m}, c_{e,m}, c_{\ell,m}\) are constant coefficients; \(S_m, E_m, L_m\) are the number of signomial, exponential, and logarithmic terms, respectively.

As illustrated in Figure 3.1, ANTIGONE responds dynamically to exploit special structure within (MINLP). ANTIGONE falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex MINLP that rigorously bound the global solution; finds feasible solutions via local optimization; divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [14, 15].

### 1.1 Licensing and software requirements

Using GAMS/ANTIGONE requires

1. an ANTIGONE license,
2. a CPLEX license, and
3. a CONOPT or SNOPT license.

ANTIGONE performs equivalently to GAMS/GloMIQO in the case where all nonlinearities in (MINLP) are quadratic; an ANTIGONE license includes the use of GloMIQO.

### 1.2 Running GAMS/ANTIGONE

GAMS/ANTIGONE solves: NLP; MINLP; RMINLP; QCP; MIQCP; RMIQCP; CNS. If GAMS/ANTIGONE is not the default solver for these models, it can be called using the following command before the `solve` statement:

```plaintext
option nlp=antigone, minlp=antigone, rminlp=antigone;
```

### 2 GAMS/ANTIGONE Output

The log output shown below is generated using the MINLP model `cecil_13` from the MINLPLib.
ANTIGONE: Algorithms for continuous/integer Global Optimization; Version 1.0

Ruth Misener and Christodoulos A. Floudas

Computer-Aided Systems Laboratory (CASL)
Department of Chemical & Biological Engineering; Princeton University

Before Pre-processing:
840 Variables
  660 Continuous
  180 Binary
929 Equations

After Pre-processing:
520 Variables
  418 Continuous
  102 Binary
499 Equations
  291 Linear
  208 Nonconvex nonlinear
232 Nonlinear Terms
  232 Signomial
730 Possible Reformulation Linearization Technique (RLT) equations
34 RLT Equations Added Outright to Formulation

Constituent Libraries:
CPLEX Solving relaxations
CONOPT Finding feasible points
LAPACK Addressing linear systems
Boost Bounding Intervals

Time (s) Nodes explored Nodes remaining Best possible Best found Relative Gap

<table>
<thead>
<tr>
<th>Time</th>
<th>Nodes explored</th>
<th>Nodes remaining</th>
<th>Best possible</th>
<th>Best found</th>
<th>Relative Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>1</td>
<td>1</td>
<td>-1.156565e+05</td>
<td>-1.157e+05</td>
<td>+1.032e-03</td>
</tr>
<tr>
<td>134</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.337e-04</td>
</tr>
<tr>
<td>202</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.334e-04</td>
</tr>
<tr>
<td>258</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.140e-06</td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<table>
<thead>
<tr>
<th>Time</th>
<th>Nodes explored</th>
<th>Nodes remaining</th>
<th>Best possible</th>
<th>Best found</th>
<th>Relative Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>341</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.091e-06</td>
</tr>
<tr>
<td>413</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.011e-06</td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<table>
<thead>
<tr>
<th>Time</th>
<th>Nodes explored</th>
<th>Nodes remaining</th>
<th>Best possible</th>
<th>Best found</th>
<th>Relative Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>483</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.001e-06</td>
</tr>
<tr>
<td>571</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+4.001e-06</td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<table>
<thead>
<tr>
<th>Time</th>
<th>Nodes explored</th>
<th>Nodes remaining</th>
<th>Best possible</th>
<th>Best found</th>
<th>Relative Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>640</td>
<td>1</td>
<td>1</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+3.880e-06</td>
</tr>
</tbody>
</table>

Solving MILP relaxation at tree level 0

<table>
<thead>
<tr>
<th>Time</th>
<th>Nodes explored</th>
<th>Nodes remaining</th>
<th>Best possible</th>
<th>Best found</th>
<th>Relative Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>702</td>
<td>1</td>
<td>0</td>
<td>-1.157e+05</td>
<td>-1.157e+05</td>
<td>+1.000e-06</td>
</tr>
</tbody>
</table>

Termination Status : Global minimum
Best Feasible Point: -1.156565e+05
Best Possible Point: -1.156566e+05
Relative Gap: +1.000000e-06

Algorithm analysis:

- 0 Nodes explored
- 0 Nodes remaining
- 0 Maximum tree depth
- 183 Cutting Planes (183 Globally Valid)
- 183 Signomial

702.38 Total time (CPU s)

- 0.07 Pre-processing
- 698.56 Solving MILP relaxations
- 0.95 Searching for feasible solutions
- 2.79 Variable bounds tightening
- 2.31 OBBT
- 1.48 FBBT (0.13 EC; 0.92 RLT; 0.00 Factoring)
- 0.00 Branching
- 0.00 Reliability branching

---

3 Summary of ANTIGONE Options

3.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_opt_tol</td>
<td>absolute stopping tolerance</td>
<td>GAMS optca</td>
</tr>
<tr>
<td>dumpsolutions</td>
<td>name of solutions index gdx file for writing alternate solutions</td>
<td></td>
</tr>
<tr>
<td>max_number_nodes</td>
<td>node limit</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td>max_time</td>
<td>resource limit</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>readparams</td>
<td>read secondary option file in ANTIGONE syntax</td>
<td></td>
</tr>
<tr>
<td>rel_opt_tol</td>
<td>relative stopping tolerance</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td>trydual</td>
<td>call CONOPT or SNOPT to produce duals</td>
<td>5</td>
</tr>
</tbody>
</table>

3.2 Options for Solving the MILP Relaxations

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cplex_optfile</td>
<td>read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve</td>
<td>1e-4</td>
</tr>
<tr>
<td>cut_generation_epsilon</td>
<td>absolute violation threshold for separating hyperplanes</td>
<td>100</td>
</tr>
<tr>
<td>nominal_time_limit</td>
<td>nominal time limit for solving MILP subproblems</td>
<td>3</td>
</tr>
<tr>
<td>populate_solution_pool</td>
<td>emphasis on generating starting points</td>
<td></td>
</tr>
</tbody>
</table>

3.3 Options for Finding Feasible Solutions

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>feas_soln_time_limit</td>
<td>time limit (s) for an NLP solve</td>
<td>30</td>
</tr>
</tbody>
</table>
feas_tolerance | absolute feasibility tolerance | 1e-6
nlp_solver | use CONOPT or SNOPT to find feasible solutions | conopt

### 3.4 Options for Branching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching_bounds_push_away</td>
<td>branch a minimum fraction away from the variable bounds</td>
<td>0.1</td>
</tr>
<tr>
<td>branching_weight</td>
<td>branch on a convex combination of midpoint and solution</td>
<td>0.25</td>
</tr>
<tr>
<td>num_reliability_tests</td>
<td>number of strong branching initialization tests</td>
<td>8</td>
</tr>
<tr>
<td>reliability_branching</td>
<td>heuristic choice for building reliable pseudocosts</td>
<td>error</td>
</tr>
<tr>
<td>reliability_branching_mu</td>
<td>score parameter for building reliability</td>
<td>0.15</td>
</tr>
<tr>
<td>use_reliability_branching</td>
<td>use reliability branching?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.5 Options for Bounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbbt_improvement_bound</td>
<td>bounds reduction improvement threshold needed to exit FBBT loop</td>
<td>0.999</td>
</tr>
<tr>
<td>max_fbbt_iterations</td>
<td>maximum number of FBBT iterations</td>
<td>50</td>
</tr>
<tr>
<td>max_obbt_iterations</td>
<td>maximum number of OBBT iterations</td>
<td>30</td>
</tr>
<tr>
<td>max_time_each_obbt</td>
<td>time limit (s) for each OBBT LP</td>
<td>10</td>
</tr>
<tr>
<td>obbt_improvement_bound</td>
<td>bounds reduction improvement threshold</td>
<td>0.95</td>
</tr>
<tr>
<td>use_obbt</td>
<td>use optimality-based bounds tightening?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.6 Options for Logging to the Console

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_freq</td>
<td>how often should we log progress to the console?</td>
<td>5</td>
</tr>
<tr>
<td>logging_level</td>
<td>logging information level</td>
<td>-1</td>
</tr>
<tr>
<td>print_options</td>
<td>print the option parameter choices used in a single run?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.7 Options for Addressing Special Structure

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptive_add_rlt</td>
<td>use the dynamic approach to adaptively determine deep RLT cuts?</td>
<td>1</td>
</tr>
<tr>
<td>adaptive_add_rlt_tree_depth</td>
<td>tree depth for heuristic that adaptively determines deep RLT cuts</td>
<td>3</td>
</tr>
<tr>
<td>add_bilinear_terms</td>
<td>allow addition of nonconvex bilinear terms to generate deep RLT cuts</td>
<td>1</td>
</tr>
<tr>
<td>convexity_cuts</td>
<td>derive convexity-based separating cuts for multivariable terms?</td>
<td>1</td>
</tr>
<tr>
<td>dominant_ec_only</td>
<td>add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations?</td>
<td>1</td>
</tr>
<tr>
<td>eigenvector_projections</td>
<td>use eigenvector projections as additional cuts?</td>
<td>1</td>
</tr>
<tr>
<td>eigenvector_projection_partitioning</td>
<td>allow partitioning on eigenvector projections?</td>
<td>1</td>
</tr>
<tr>
<td>low_dim_edge_concave_agg</td>
<td>use low-dimension edge-concave aggregations?</td>
<td>1</td>
</tr>
<tr>
<td>max_partitioned_quantities</td>
<td>number of partitioned quantities</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>max_rlt_cuts</td>
<td>maximum number of violated RLT cuts to add before resolving the relaxation?</td>
<td>100</td>
</tr>
<tr>
<td>naive_add_ec</td>
<td>naively integrate all low-dimension edge-concave aggregations into relaxations?</td>
<td>0</td>
</tr>
<tr>
<td>naive_add_rlt</td>
<td>naively add all RLT cuts to the relaxations?</td>
<td>0</td>
</tr>
<tr>
<td>number_of_partitions</td>
<td>how many partitions per variable?</td>
<td>1</td>
</tr>
<tr>
<td>partitioning_scheme</td>
<td>Partitioning scheme can be linear or logarithmic</td>
<td>linear</td>
</tr>
<tr>
<td>piecewise_linear_partitions</td>
<td>use piecewise-linear partitioning?</td>
<td>0</td>
</tr>
<tr>
<td>rlt</td>
<td>find RLT variable/equation and equation/equation pairs?</td>
<td>1</td>
</tr>
<tr>
<td>use_alpha_bb</td>
<td>apply globally-valid alphaBB cuts to tighten a node relaxation</td>
<td>1</td>
</tr>
<tr>
<td>use_edge_concave_dynamic</td>
<td>apply locally-valid edge-concave cuts to tighten a node relaxation</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4 Detailed Descriptions of ANTIGONE Options

**abs_opt_tol (real):** absolute stopping tolerance  
(default = GAMS optca)

**adaptive_add_rlt (integer):** use the dynamic approach to adaptively determine deep RLT cuts?  
In the first few levels of the branch-and-bound tree, query the RLT equations after solving an initial relaxation. Add violated equations to the relaxation and resolve. Track the most commonly-violated equations and include those cuts in later nodes.  
(default = 1)

**adaptive_add_rlt_tree_depth (integer):** tree depth for heuristic that adaptively determines deep RLT cuts  
To the specified tree depth, solve the relaxation of a node twice if RLT equations are violated. After this depth, automatically add the most commonly violated cuts to the solution of each node  
Range: [1, 100]  
(default = 3)

**add_bilinear_terms (integer):** allow addition of nonconvex bilinear terms to generate deep RLT cuts  
(default = 1)

**branching_bounds_push_away (real):** branch a minimum fraction away from the variable bounds  
Range: [0, 0.5]  
(default = 0.1)

**branching_weight (real):** branch on a convex combination of midpoint and solution  
The branching weight specifies the emphasis on the midpoint of a variable, so larger branching weights imply branching closer to the center of a variable range.  
Range: [0, 1]  
(default = 0.25)

**convexity_cuts (integer):** derive convexity-based separating cuts for multivariable terms?  
(default = 1)

**cplex_optfile (string):** read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve  
Gain direct access to the GAMS/CPLEX options. Specifying an options file allows, for example, the possibility of running the CPLEX subsolver with multiple threads. The value of the string should match the name of the GAMS/CPLEX options file.

**cut_generation_epsilon (real):** absolute violation threshold for separating hyperplanes
Absolute violation threshold to generate separating hyperplanes for convex multivariable terms
Range: $[1e^{-7}, 10]$  
(default = $1e^{-4}$)

**dominant_ec_only (integer):** add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations?
(default = 1)

**dump solutions (string):** name of solutions index gdx file for writing alternate solutions
The GDX file specified by this option will contain a set call `index` that contains the names of GDX files with the individual solutions. For details see example model dumpsol in the GAMS Test Library.

**eigenvector_projections (integer):** use eigenvector projections as additional cuts?
(default = 1)

**eigenvector_projection_partitioning (integer):** allow partitioning on eigenvector projections?
(default = 1)

**fbbt_improvement_bound (real):** bounds reduction improvement threshold needed to exit FBBT loop
Range: $[0, 1]$  
(default = 0.999)

**feas_soln_time_limit (real):** time limit (s) for an NLP solve
(default = 30)

**feas_tolerance (real):** absolute feasibility tolerance
(default = $1e^{-6}$)

**logging_freq (real):** how often should we log progress to the console?
Wait at least the specified time in seconds before next output to the console
(default = 5)

**logging_level (integer):** logging information level
Log to the console at the specified level (-1: default; 0: minimal logging; 3: extensive logging)
Range: $[-1, 3]$  
(default = -1)

-1 minimal plus warnings
0 minimal
1 entering info
2 updating info
3 includes Cplex updates

**low_dim_edge_concave_agg (integer):** use low-dimension edge-concave aggregations?
(default = 1)

**max_fbbt_iterations (integer):** maximum number of FBBT iterations
Range: $[1, 100]$  
(default = 50)

**max_number_nodes (integer):** node limit
(default = GAMS nodlim)

**max_obbt_iterations (integer):** maximum number of OBBT iterations
Range: [1, 100]
(default = 30)

**max_partitioned_quantities (integer):** number of partitioned quantities

Range: [0, 50]
(default = 0)

**max_rlt_cuts (integer):** maximum number of violated RLT cuts to add before resolving the relaxation?

Range: [1, 1000]
(default = 100)

**max_time (real):** resource limit

(default = GAMS reslim)

**max_time_each_obbt (real):** time limit (s) for each OBBT LP

Range: [1, 100]
(default = 10)

**naive_addEc (integer):** naively integrate all low-dimension edge-concave aggregations into relaxations?

(default = 0)

**naive_add_rlt (integer):** naively add all RLT cuts to the relaxations?

(default = 0)

**nlp_solver (string):** use CONOPT or SNOPT to find feasible solutions

(default = conopt)

- conopt Conopt
- snopt Snopt

**nominal_time_limit (real):** nominal time limit for solving MILP subproblems

Nominal time limit for solving MILP subproblems. Terminate long-running MILP subproblems over this time limit once they reach an integer feasible point

Range: [0.1, 1000]
(default = 100)

**number_of_partitions (integer):** how many partitions per variable?

Range: [0, 16]
(default = 1)

**num_reliability_tests (integer):** number of strong branching initialization tests

Range: [1, 100]
(default = 8)

**obbt_improvement_bound (real):** bounds reduction improvement threshold

Bounds reduction improvement threshold needed to exit OBBT loop This parameter also determines whether to continue obbt in child; if the parent bound improvement is less than this threshold, then child node won’t try OBBT

Range: [0, 1]
(default = 0.95)

**partitioning_scheme (string):** Partitioning scheme can be linear or logarithmic

Linear partitioning uses a number of binary variables linear in the number of partitions while logarithmic partitioning uses a number of binary variables logarithmic in the number of breakpoints. Linear partitioning
tends to be numerically favorable for a few breakpoints while logarithmic partitioning is better for a larger number of breakpoints.

(default = linear)

linear Linear partitioning
logarithmic Logarithmic partitioning

piecewise_linear_partitions (integer): use piecewise-linear partitioning?

(default = 0)

populate_solution_pool (integer): emphasis on generating starting points

Emphasis on generating many starting points for NLP solves using the CPLEX solution pool feature. Larger number implies more starting points.

Range: [0, 4]

(default = 3)

print_options (integer): print the option parameter choices used in a single run?

(default = 1)

readparams (string): read secondary option file in ANTIGONE syntax

reliability_branching (string): heuristic choice for building reliable pseudocosts

(default = error)

error Max Error Branching
forward Forward branching
reverse Reverse branching

reliability_branching_mu (real): score parameter for building reliability

Range: [0, 1]

(default = 0.15)

rel_opt_tol (real): relative stopping tolerance

(default = GAMS optcr)

rlt (integer): find RLT variable/equation and equation/equation pairs?

(default = 1)

trydual (real): call CONOPT or SNOPT to produce duals

Spend the specified amount of time in seconds or less in producing a dual solution by calling CONOPT or SNOPT.

Range: [0, maxdouble]

(default = 5)

use_alpha_bb (integer): apply globally-valid alphaBB cuts to tighten a node relaxation

(default = 1)

use_edge_concave_dynamic (integer): apply locally-valid edge-concave cuts to tighten a node relaxation

(default = 1)

use_obbt (integer): use optimality-based bounds tightening?

(default = 1)

use_reliability_branching (integer): use reliability branching?

(default = 1)
5 ANTIGONE Algorithmic Features

As illustrated in Figure 3.1, the primary algorithmic features in ANTIGONE are reformulating model input, elucidating special structure, and branch-and-bound global optimization [27–31].

5.1 Reformulating Model Input

As illustrated in Figure 3.2, ANTIGONE transforms a factorable programming tree into a flattened expression tree to capitalize on the development of tight convex underestimators for specific classes of nonlinear terms. ANTIGONE extends the efficacy of hybrid strategies by meaningfully integrating mutually reinforcing operator- and term-based strategies [7, 16, 29]. This approach reformulates towards multivariable terms with specialized underestimators while maintaining a tree-like representation of powers that cannot be distributed and convex operators that can be exploited by dynamic cut generation.

5.2 Elucidating Special Structure

After reformulating the user-defined MINLP, ANTIGONE detects special mathematical structure that it will exploit in the branch-and-cut phase (Section Branch-and-Bound Global Optimization). The types of special structure that ANTIGONE considers are: reformulation-linearization technique (RLT) equations; convexity/concavity; edge-convexity/edge-concavity; αBB relaxations; term-specific underestimators [27–31].

- **RLT** multiplies every pairwise combination of: variables; nonlinear terms; equations [5, 20, 28, 29, 31, 35–39]. ANTIGONE saves the combinations that do not introduce new terms into the model formulation and updates these equations at every node of the branch-and-cut tree. Special RLT equations are added directly to the model formulation; other RLT equations are used as cutting planes and integrated into the feasibility-based bounds tightening routines.

- **Convexity/Concavity** permits the easy generation of a cutting plane at a point \( \bar{x} \):

\[
\begin{align*}
    f(x) &\geq f(\bar{x}) + f'(\bar{x}) \cdot (x - \bar{x}) \quad \text{(convex)} \\
    f(x) &\leq f(\bar{x}) + f'(\bar{x}) \cdot (x - \bar{x}) \quad \text{(concave)}
\end{align*}
\]

Based on interval arithmetic, terms and multi-term expressions are labeled as always/sometimes/never convex/concave; this information is used in the branch-and-cut phase.

- **Edge-Convexity/Edge-Concavity** implies a vertex polyhedral envelope; ANTIGONE labels terms and multi-term expressions as always/sometimes/never edge-convex/edge-concave with a simple interval arithmetic test [26, 40–42].

- **αBB** underestimators convexify an expression with univariate quadratics [2–4, 13, 24]; ANTIGONE uses αBB to relax aggregates of bilinear terms.
• **Term-Specific Underestimators** are diagrammed in Figure 3.3; their implementation is based on work available in the open literature [10–12, 17, 19, 21–23, 25–27, 32, 40, 41, 43].

### 5.3 Branch-and-Bound Global Optimization

After the reformulation and special structure detection phases, ANTIGONE initiates a branch-and-cut global optimization algorithm that generates tight convex underestimators, dynamically generates separating hyperplanes, bounds the variables [2–6, 8, 9, 18, 33, 34, 38, 39, 44]; branches on the search space [1, 6], and finds feasible solutions.

### Bibliography


1 Introduction

The Branch-And-Reduce Optimization Navigator (BARON) is a GAMS solver for the global solution of nonlinear (NLP) and mixed-integer nonlinear programs (MINLP).

While traditional NLP and MINLP algorithms are guaranteed to converge only under certain convexity assumptions, BARON implements deterministic global optimization algorithms of the branch-and-bound type that are guaranteed to provide global optima under fairly general assumptions. These include the existence of finite lower and upper bounds on nonlinear expressions in the NLP or MINLP to be solved.
BARON implements algorithms of the branch-and-bound type enhanced with a variety of constraint propagation, interval analysis, and duality techniques for reducing ranges of variables in the course of the algorithm. Rigorous relaxations are constructed by enlarging the feasible region and/or underestimating the objective function.

Parts of the BARON software were created at the University of Illinois at Urbana-Champaign. The algorithms implemented in the software, the theory behind them, and some related applications are (partly) described in [1–36].

1.1 Licensing and software requirements

In order to use GAMS/BARON, users will need to have a GAMS/BARON license. The software includes the Coin solvers CLP and IPOPT for solving BARON’s linear programming (LP) and nonlinear programming (NLP) subproblems, respectively. Licensed GAMS LP and NLP solvers are optional and usually expedite convergence. Current valid LP subsolvers are CLP, CPLEX, and XPRESS. Current valid NLP subsolvers are CONOPT, IPOPT, MINOS, and SNOPT.

By default, GAMS/BARON will attempt to use CPLEX as the LP solver and select an NLP solver automatically. The user can use the options LPSol and NLPSol to specify the LP or NLP solver. If the user does not have a license for the default or user-specified LP solver, BARON will switch to CLP. If the user does not have a license for the user-specified NLP solver, BARON will automatically select a licensed NLP solver and may default to IPOPT if no other NLP solver is available. BARON can be used without a local NLP solver by setting DoLocal and and NumLoc to 0.

1.2 Running GAMS/BARON

BARON is capable of solving models of the following types: LP, MIP, RMIP, NLP, DNLP, RMINLP, and MINLP. If BARON is not specified as the default solver for these models, it can be invoked by issuing the following command before the solve statement:

```
option <modeltype>=baron;
```

where `<modeltype>` stands for LP, MIP, RMIP, QCP, MIQCP, RMICQP, CNS, NLP, DNLP, MINLP, or RMINLP.

2 Model requirements

In order to achieve convergence to global optimality, additional model constraints may be required. The additional constraints may speed up solver time and increase the probability of success.

2.1 Variable and expression bounds

All nonlinear expressions in the mathematical program to be solved must be bounded below and/or above. It is important that finite lower and upper bounds be provided by the user on all problem variables. Note that providing finite bounds on variables alone may not suffice to guarantee finite bounds on nonlinear expressions arising in the model.

For example, consider the term 1/x for x ∈ [0,1], which has finite variable bounds, but is unbounded. It is important to provide bounds for problem variables that guarantee that the problem functions are finitely-valued. If the user model does not include variable bounds that guarantee that all nonlinear expressions are finitely-valued, BARON will attempt to infer appropriate bounds from problem constraints. If this step fails, global optimality of the solutions provided is not guaranteed. Occasionally, because of the lack of bounds no numerically stable lower bounding problems can be constructed, in which case BARON may terminate.

See section Some BARON features on how to specify variable bounds.

2.2 Allowable nonlinear functions

In addition to multiplication and division, GAMS/BARON can handle nonlinear functions that involve exp(x), ln(x), x^α for real α, β^x for real β, x^y, and |x|. Currently, there is no support for other functions, including the trigonometric functions sin(x), cos(x), etc.
3  BARON output

3.1  BARON log output

The log output below is obtained for the MINLP model gear.gms from the GAMS model library using absolute and relative optimality tolerances of 1e-6.

===========================================================================
If you use this software, please cite:
BARON is a product of The Optimization Firm, LLC. http://www.minlp.com/
Parts of the BARON software were created at the University of Illinois at Urbana-Champaign.
===========================================================================
This BARON run may utilize the following subsolver(s)
For LP/MIP: ILOG CPLEX
For NLP: MINOS, SNOPT, GAMS external NLP, COIN IPOPT with MUMPS and METIS
===========================================================================
Starting solution is feasible with a value of 36.1767610000
Doing local search
Preprocessing found feasible solution with value 1.01580224842
Solving bounding LP
Starting multi-start local search
Preprocessing found feasible solution with value 1.00209253056
Done with local search
===========================================================================
<table>
<thead>
<tr>
<th>Iteration</th>
<th>Open nodes</th>
<th>Total time</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00209</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00209</td>
</tr>
<tr>
<td>*</td>
<td>5</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00117</td>
</tr>
<tr>
<td>*</td>
<td>8</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00059</td>
</tr>
<tr>
<td>*</td>
<td>17</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00018</td>
</tr>
<tr>
<td>*</td>
<td>22</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00001</td>
</tr>
<tr>
<td>*</td>
<td>35</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>000:00:00</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

Cleaning up

*** Normal completion ***

LP subsolver time: 000:00:00, in seconds: 0.01
MIP subsolver time: 000:00:00, in seconds: 0.01
NLP subsolver time: 000:00:00, in seconds: 0.01
Cutting time: 000:00:00, in seconds: 0.00
All other time: 000:00:00, in seconds: 0.03
Total time elapsed: 000:00:00, in seconds: 0.06
on parsing: 000:00:00, in seconds: 0.00
on preprocessing: 000:00:00, in seconds: 0.02
on navigating: 000:00:00, in seconds: 0.01
on relaxed: 000:00:00, in seconds: 0.02
The solver first tests feasibility of the user-supplied starting point. This point is found to be feasible with an objective function value of 36.1767610000. BARON subsequently performs a randomized local search procedure and, eventually, finds a feasible solution with an objective of 1.00209253056. Then, the iteration log provides information every 1,000,000 branch-and-bound iterations and every 30 seconds. Additionally, information is printed at the end of the root node, whenever the value of the incumbent is improved by at least $10^{-5}$, and at the end of the search. A star (*) in the first position of a line indicates that a better feasible solution was found that improves the value of previous best known solution by at least $10^{-5}$. The log fields include the iteration number, number of open branch-and-bound nodes, the CPU time taken thus far, the lower bound, and the upper bound for the problem. The log output fields are summarized below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Itn. no.</td>
<td>The number of the current iteration. A plus (+) following the iteration number denotes reporting while solving a probing (as opposed to a relaxation) subproblem of the corresponding node.</td>
</tr>
<tr>
<td>Open Nodes</td>
<td>Number of open nodes in branch-and-reduce tree.</td>
</tr>
<tr>
<td>Total Time</td>
<td>Current elapsed resource time in seconds.</td>
</tr>
<tr>
<td>Lower Bound</td>
<td>Current lower bound on the model.</td>
</tr>
<tr>
<td>Upper Bound</td>
<td>Current upper bound on the model.</td>
</tr>
</tbody>
</table>

Once the branch-and-reduce tree is searched, the best solution is isolated and a corresponding dual solution is calculated. Finally, the total number of branch-and-reduce iterations (number of search tree nodes) is reported, followed by the node where the best solution was identified (a -1 indicates preprocessing as explained in the next section on termination messages). Finally, some information is provided about the number and type of cutting planes generated during the search.

### 3.2 Termination messages, model and solver statuses

Upon normal termination, BARON will report the node where the optimal solution was found. We refer to this node as nodeopt. Associated with this node is a return code indicating the status of the solution found at nodeopt. The return code is given in the log line:

Best solution found at node: (return code)

The return codes have the following interpretation:

-3 : no feasible solution found
-2 : the best solution found was the user-supplied
-1 : the best solution was found during preprocessing
  i : the best solution was found in the i-th node of the tree
In addition to reporting nodeopt, upon termination, BARON will issue one of the following statements:

*** Normal completion ***
This is the most desirable termination status. The problem has been solved within tolerances in this case. If BARON returns a code of -3, then no feasible solution exists.

*** Heuristic termination ***
While global optimality is not guaranteed in this case, BARON will terminate with this message when (a) a feasible solution has been found and (b) the progress of lower/upper bounds satisfies the heuristic termination criterion set by the user through BARON’s DeltaTerm option.

*** User did not provide appropriate variable bounds ***
The user will need to read the BARON output (in file sum.dat in the scratch directory, use GAMS parameter keep=1 to prevent the automatic removal of this directory) for pointers to variables and expressions with missing bounds. The model should be modified in order to provide bounds for variables and intermediate expressions that make it possible for BARON to construct reliable relaxations. Even though relaxation bounds are printed on the screen to give the user a feeling for convergence, these bounds may not be valid for the problem at hand. This message is followed by one of the following two messages:

*** Infeasibility is therefore not guaranteed ***
This indicates that, because of missing bounds, no feasible solution was found, but model infeasibility was not proven.

*** Globality is therefore not guaranteed ***
This indicates that, because of missing bounds, a feasible solution was found, but global optimality was not proven.

*** Max. allowable nodes in memory reached ***
The user will need to increase the physical memory of the computer or change algorithmic options, such as branching and node selection rules, to reduce the size of the search tree and memory required for storage.

*** Max. allowable B&B iterations reached ***
The user will need to increase the maximum number of allowable iterations. The BARON option is MaxIter. To specify this in GAMS, one can use the NodLim option. We remark that the BARON option MaxIter overrides NodLim.

*** Max. allowable CPU time exceeded ***
The user will need to increase the maximum of allowable CPU time. The BARON option is MaxTime. To specify this in GAMS, one can use the ResLim option. We remark that the BARON option MaxTime overrides ResLim.

*** Problem is numerically sensitive ***
BARON is designed to automatically handle problems with numerical difficulties. However, for certain problems, the global optimum is numerically sensitive. This occurs, for instance, when the objective function value varies significantly over small neighborhoods of points that are strictly outside the feasible region but nonetheless feasible within numerical tolerances. When BARON returns this message, the "Best possible" reported on the objective is likely correct.

*** Search interrupted by user ***
The run was interrupted by the user (Ctrl-C).

*** Insufficient memory for data structures ***
More memory is needed to set up the problem data structures. The user will need to increase the physical memory available on the computer in order to accommodate problems of this size.

*** A potentially catastrophic access violation just took place ***
In the unlikely event of a memory access violation, BARON will terminate the search and return the best known solution. Please report problems that lead to this condition to Nick Sahinidis (niksah@minlp.com).

4 Some BARON features

The features described in this section rely on options that are further detailed in the next section. The user may also wish to consult the Tawarmalani-Sahinidis book \(^1\) for more details on BARON features and illustrations of their use. Further details are offered in the publications cited at the end of this document.

4.1 No starting point is required

In contrast to many NLP algorithms that require a feasible starting point, a starting point is not required for BARON. A user may optionally provide a starting point for all or even some of the problem variables. BARON will judiciously initialize any variables that are not initialized by the user. Even when the problem functions cannot be evaluated at a user-provided starting point, BARON is still capable of carrying out its global search.

For problems for which GAMS compilation is aborted because the nonlinear functions cannot be evaluated at the starting point, the user can use the following commands before the \texttt{SOLVE} statement:

\begin{verbatim}
MaxExecError = 100000;
option sys12 = 1;
\end{verbatim}

The first command asks GAMS to continue compilation for as many as \texttt{MaxExecError} execution errors. The \texttt{sys12} option will pass the model to the BARON despite the execution errors. Even though the starting point is bad in this case, BARON is capable of carrying out its global search.

4.2 Finding a few of the best or all feasible solutions

BARON offers a facility, through its \texttt{NumSol} option, to find the best few, or even all feasible, solutions to a model. The development of this facility was motivated by combinatorial optimization problems but the facility is applicable to continuous problems as well. Even for combinatorial problems, BARON does not rely on integer cuts to find multiple solutions. Instead, it utilizes a single search tree, thus providing a computationally efficient method for finding multiple solutions. Furthermore, because BARON’s approach applies to integer as well as continuous programs, it can be used to find all feasible solutions to a system of nonlinear equality and inequality constraints.

Once a model is solved by BARON with the \texttt{NumSol} option, the solutions found can be recovered using the GAMS GDX facility. An example is provided below.

\begin{verbatim}
$eolcom !
$Ontext
  Purpose: demonstrate use of BARON option 'numsol' to obtain the best
  numsol solutions of an optimization problem in a single branch-and-bound
  search tree.

  The model solved here is a linear general integer problem with 18 feasible
  solutions. BARON is run with a request to find up to 20 solutions. The
  model solved is the same as the one solved in gamslib/icut.gms.
$Offtext

set i index of integer variables / 1 * 4 /
\end{verbatim}

variables x(i) variables
  z         objective variable

integer variable x;

x.lo(i) = 2; x.up(i) = 4; x.fx('2') = 3;   ! fix one variable
x.up('4') = 3;       ! only two values

equation obj obj definition;

* pick an objective function which will order the solutions

obj .. z =e= sum(i, power(10,card(i)-ord(i))*x(i));

model enum / all /;

* instruct BARON to return numsol solutions
$onecho > baron.opt
  numsol 20
  gdxout multsol
$offecho

enum.optfile=1; option mip=baron, limrow=0, limcol=0, optca=1e-5,
  optcr=1e-5; solve enum minimizing z using mip;

* recover BARON solutions through GDX
set sol /multsol1*multsol100/; variables xsol(sol,i), zsol(sol);
execute 'gdxmerge multsol*.gdx > %gams.scrdir%merge.%gams.scrext%';
execute_load 'merged.gdx', xsol=x, zsol=z;

option decimals=8;
display xsol.l, zsol.l;

4.3  Using BARON as a multi-start heuristic solver

To gain insight into the difficulty of a nonlinear program, especially with regard to existence of multiple local solutions,
modelers often make use of multiple local searches from randomly generated starting points. This can be easily done with
BARON’s NumLoc option, which determines the number of local searches to be done by BARON’s preprocessor. BARON
can be forced to terminate after preprocessing by setting the number of iterations to 0 through the MaxIter option. In addition
to local search, BARON’s preprocessor performs extensive reduction of variable ranges. To sample the search space for local
minima without range reduction, one would have to set to 0 the range reduction options TDo, MDo, LBTTDo, and OBTTDo.
On the other hand, leaving these options to their default values increases the likelihood of finding high quality local optima
during preprocessing. If NumLoc is set to -1, local searches in preprocessing will be done from randomly generated starting
points until global optimality is proved or MaxTime CPU seconds have elapsed.

4.4  Systematic treatment of unbounded problems

If BARON declares a problem as unbounded, it will search for and may report a vertex and direction of an unbounded ray.
In addition, BARON will report the best solution found. This will be a feasible point that is as far as possible along an
unbounded ray while avoiding numerical errors due to floating point arithmetic.
4.5 Systematic treatment of infeasible problems

If BARON declares a problem as infeasible, it has the capability to identify a subset of the constraints that are infeasible and become feasible once any one of them is eliminated. This, so-called, *irreducibly inconsistent system* (IIS) can be obtained by BARON for all types of problems handled by BARON, including linear and nonlinear, continuous and integer, convex and nonconvex, and problems with complementarity constraints. BARON’s CompIIS option can be used to identify an IIS.

As an example, consider the problem of minimizing the nonconvex function $x_1 x_3$ over the following nonconvex constrained set:

$$
\begin{align*}
e_1 : & \quad 85 + 0.006x_2x_5 + 0.0006x_1x_4 - 0.002x_3x_5 \leq 92 \\
e_2 : & \quad 0.8x_2x_5 + 0.003x_1x_2 + 0.002x_3^2 = 110 \\
e_3 : & \quad 9 + 0.005x_3x_5 + 0.001x_1x_3 + 0.002x_3x_4 \leq 25 \\
& \quad 78 \leq x_1 \leq 102 \\
& \quad 33 \leq x_2 \leq 45 \\
& \quad 27 \leq x_i \leq 45, \quad i = 3, \ldots, 5
\end{align*}
$$

When this problem is solved with CompIIS equal to 1, BARON provides the following result:

IIS contains 1 row and 3 columns as follows:
- $e_2$ Upper
- $x_1$ Lower
- $x_2$ Lower
- $x_5$ Lower

The IIS consists of the lower bounds of variables $x_1$, $x_2$, and $x_3$, along with the $\leq$ part of the equality constraint $e_2$. This suggests that constraint $e_2$ and the entire model can be made feasible by lowering the lower bound of any of the three variables that are part of the IIS, whereas modifying the bounds of $x_3$ would not make the model feasible.

4.6 Handling of complementarity constraints

Complementarity relationships of the type $f(x)g(x) = 0$ are automatically recognized and exploited algorithmically by BARON. The functions $f$ and $g$ may be univariate or multivariate, linear or nonlinear, convex or nonconvex, in terms of continuous and/or integer variables, and may be subject to additional constraints in the model. These complementarity relationships can be inferred by BARON even when implied by problem constraints and variable bounds. As a result, BARON can solve general mathematical programs with equilibrium constraints (MPECs). This class of problems includes the classical linear complementarity problem

$$(LCP): \text{Find } z \geq 0 \text{ and } q \text{ such that } Mz + q \geq 0 \text{ and } z^t(Mz + q) = 0$$

as well as the more general mixed complementarity problem

$$(MCP): \text{Given a function } f : \mathbb{R}^n \to \mathbb{R}^n \text{ and bounds } l, u \in \mathbb{R}^n \text{ with } \bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}, \text{ find } z \in \mathbb{R}^n \text{ and } w, v \in \mathbb{R}_+^n \text{ such that }$$

$$f(z) = w - v, \quad l \leq z \leq u, \quad (z - l)^t w = 0, \quad (u - z)^t v = 0.$$
5 The BARON options

The BARON options allow the user to control termination tolerances, branching and relaxation strategies, heuristic local search options, and output options as detailed in this section.

Many options can also be set in the GAMS model. The most relevant GAMS options are ResLim, NodLim, OptCA, OptCR, OptFile, and CutOff. The IterLim option is not implemented. Specify BARON iterations, the user can set the MaxIter option, which is equivalent to the GAMS option NodLim.

Additionally, a BARON Options file can be provided. See the The Solver Option File section for general use of solver option files.

For Tree management options, users can specify separate branching priorities for any discrete and continuous variables using GAMS Dot Options. To specify variable branching priorities, one specifies

\[(variable).prior\text{((value))}\]

in the BARON options file, where \((value)\) can be any non-negative real value. A value of zero will result in no branching on the corresponding variable.

In contrast to BARON, GAMS priorities are assigned in such a fashion that a larger value implies a lower priority. BARON and GAMS variable priorities can be thought of as being approximately related by

BARON priority = 1 / GAMS priority.

5.1 Termination options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsConFeasTol</td>
<td>Absolute constraint feasibility tolerance</td>
<td>1e-5</td>
</tr>
<tr>
<td></td>
<td>This tolerance is used for general constraints and variable bounds.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AbsConFeasTol must be (\geq 1e-12).</td>
<td></td>
</tr>
<tr>
<td>AbsIntFeasTol</td>
<td>Absolute integer feasibility tolerance</td>
<td>1e-5</td>
</tr>
<tr>
<td></td>
<td>All integer variable values must satisfy this tolerance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AbsIntFeasTol must be (\geq 1e-12).</td>
<td></td>
</tr>
<tr>
<td>BoxTol</td>
<td>Box elimination tolerance.</td>
<td>1e-8</td>
</tr>
<tr>
<td></td>
<td>Boxes will be eliminated if smaller than this tolerance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BoxTol must be (\geq 1e-12).</td>
<td></td>
</tr>
<tr>
<td>ConTol</td>
<td>Constraint satisfaction tolerance.</td>
<td>1e-5</td>
</tr>
<tr>
<td>CutOff</td>
<td>Eliminate solutions that are no better than this value</td>
<td>GAMS CutOff</td>
</tr>
<tr>
<td></td>
<td>Can also be used with the GAMS model attribute option CutOff.</td>
<td></td>
</tr>
<tr>
<td>DeltaA</td>
<td>Absolute improvement for insufficient progress termination</td>
<td>(\infty)</td>
</tr>
<tr>
<td></td>
<td>(\Delta A (\delta_a)) must be (\geq 1e-12).</td>
<td></td>
</tr>
<tr>
<td>DeltaR</td>
<td>Relative improvement for insufficient progress termination</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(\Delta R (\delta_r)) must be (\geq 1e-12).</td>
<td></td>
</tr>
<tr>
<td>DeltaT</td>
<td>Time interval for insufficient progress termination</td>
<td>-100</td>
</tr>
<tr>
<td></td>
<td>If DeltaTerm is set to 1, BARON will terminate if insufficient progress is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>made over DeltaT ((\delta_t)) consecutive seconds. If (\delta_t) is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set to a non-positive quantity,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BARON will automatically set (\delta_t) equal to (-\delta_t) times the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPU time taken till the end of root node processing. DeltaT can take any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>real value.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>DeltaTerm</td>
<td>Indicates whether insufficient progress termination is on or off</td>
<td>0</td>
</tr>
<tr>
<td>Users have the option to request BARON to terminate if insufficient progress is made over $\Delta T (\delta_t)$ consecutive seconds. Progress is measured using the absolute and relative improvement thresholds $\Delta A (\delta_a)$ and $\Delta R (\delta_r)$. Termination will occur if, over a period of $\delta_t$ consecutive seconds, the value of the best solution found by BARON is not improved by at least an absolute amount $\delta_a$ or an amount equal to $\delta_r$ times the value of the incumbent at time $t-\delta_t$. This termination condition is enforced after processing the root node and only after a feasible solution has been obtained. Do not enforce this termination condition 0: do not enforce this termination condition 1: terminate if progress is insufficient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EpsA</td>
<td>Absolute termination tolerance. BARON terminates if $</td>
<td>U-L</td>
</tr>
<tr>
<td>EpsR</td>
<td>Relative termination tolerance. BARON terminates if $</td>
<td>U-L</td>
</tr>
<tr>
<td>FirstFeas</td>
<td>Changes the search for first numsol solutions. If set to 1, BARON will terminate once it finds NumSol feasible solutions, irrespective of solution quality. 0: search for the best NumSol feasible solutions 1: find NumSol solutions irrespective of solution quality</td>
<td>0</td>
</tr>
<tr>
<td>IntTol</td>
<td>Integritiy satisfaction tolerance. Solution Distance Separation distance between solutions. This option is used in conjunction with NumSol. For combinatorial optimization problems, feasible solutions are isolated. For continuous problems, feasible solution points within an $L_\infty$ distance that does not exceed IsolTol will be treated as identical by BARON. IsolTol must be $\geq 1e^{-12}$.</td>
<td>1e-6</td>
</tr>
<tr>
<td>ISolTol</td>
<td>Solution Distance Separation distance between solutions. This option is used in conjunction with NumSol. For combinatorial optimization problems, feasible solutions are isolated. For continuous problems, feasible solution points within an $L_\infty$ distance that does not exceed IsolTol will be treated as identical by BARON. IsolTol must be $\geq 1e^{-12}$.</td>
<td>1e-4</td>
</tr>
<tr>
<td>MaxIter</td>
<td>Maximum number of branch-and-reduce iterations allowed. Setting MaxIter to 0 will force BARON to terminate after root node preprocessing. Setting MaxIter to 1 will result in termination after the solution of the root node. MaxIter must be $\geq -1$, where -1 implies unlimited.</td>
<td>GAMS NodeLim</td>
</tr>
<tr>
<td>MaxTime</td>
<td>Maximum CPU time (in seconds) allowed. MaxTime must be -1 or $&gt; 0$, where -1 implies unlimited.</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>NumSol</td>
<td>Number of feasible solutions to be found. Solutions found will be listed in the results file. As long as NumSol $\neq -1$, these solutions will be sorted from best to worse. If NumSol is set to -1, BARON will search for all feasible solutions to the given model and print them, in the order in which they are found, in the results file. NumSol must be $\geq -1$.</td>
<td>1</td>
</tr>
<tr>
<td>RelConFeasTol</td>
<td>Relative constraint feasibility tolerance. This tolerance is used for general constraints and variable bounds. RelConFeasTol must be $\geq 0$.</td>
<td>0</td>
</tr>
<tr>
<td>RelIntFeasTol</td>
<td>Relative integer feasibility tolerance. All integer variable values must satisfy this tolerance. RelIntFeasTol must be $\geq 0$.</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.2 Relaxation options
Option | Description | Default
--- | --- | ---
mipgap | Relative optimality tolerance for MIP relaxations | 0.05
mipnodes | Number of nodes allowed in the MIP relaxation solution | 1000
miprel | Switch to use MIP relaxations | 1
multsize | Maximum allowable size of multilinear | 2
multrel | Rounds of cuts of multilinears | 4
nlpdolin | Linearization option for relaxation | 1
| Linearization option for relaxation. A value of 0 will result in the use of nonlinear relaxations whenever possible. This option should be avoided. It is offered as an alternative for hard problems but may lead to incorrect results depending on the performance of the local search solver for the problem at hand. The default value of 1 is to use a linear programming relaxation, which represents the most reliable approach under BARON. 0 use nonlinear relaxation whenever possible 1 use linear programming relaxation
Nouter1 | Number of outer approximators of convex univariant functions. Nouter1 must be \( \geq 0 \). | 4
NoutIter | Number of rounds of cutting plane generation at LP relaxation. NoutIter must be a \( \geq 0 \). | 4
NoutIterMip | Number of outer approximation rounds with MIP relaxation. | 4
NoutPerVar | Number of outer approximations per variable NoutPerVar must be \( \geq 0 \). | 4
OutGrid | Number of grid points per variable for convex multivariate approximators of BARON’s CONVEX EQUATIONS. OutGrid must be a \( \geq 0 \). | 20

5.3 Range reduction options

Option | Description | Default
--- | --- | ---
LBTTDo | Linear-feasibility-based range reduction option (poor man’s LPs) 0 no range reduction based on feasibility. 1 range reduction done based on feasibility. | 1
MDo | Marginals-based reduction option 0 no range reduction based on marginals. 1 range reduction done based on marginals. | 1
OBTTDo | Optimality based tightening option 0 no range reduction based on optimality. 1 range reduction done based on optimality. | 1
PDO | Number of probing problems allowed -2 automatically decided by BARON. -1 probing on all numbranch variables. 0 no range reduction by probing. n probing on n variables. | -2
TDo | Nonlinear-feasibility-based range reduction option (poor man’s NLPs) 0 no bounds tightening is performed. 1 bounds tightening is performed. | 1

5.4 Tree management options

Option | Description
--- | ---
### Branching Point Selection Strategy

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrPtStra</td>
<td>Branching point selection strategy</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 BARON’s dynamic strategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 w-branching</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 bisection-branching</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 convex combination of 1 and 2</td>
<td></td>
</tr>
</tbody>
</table>

### Branching Variable Selection Strategy

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrVarStra</td>
<td>Branching variable selection strategy</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 BARON’s dynamic strategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 largest violation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 longest edge</td>
<td></td>
</tr>
</tbody>
</table>

### Node Selection

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeSel</td>
<td>Specifies the node selection rule to be used for exploring the search tree.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Specifies the node selection rule to be used for exploring the search tree.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 BARON’s mixed selection scheme</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 best bound</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 last in first out [LIFO]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 minimum infeasibility</td>
<td></td>
</tr>
</tbody>
</table>

### 5.5 Local Search Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoLocal</td>
<td>Local search option for upper bounding</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 no local search is done during upper bounding</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 BARON’s dynamic local search decision rule</td>
<td></td>
</tr>
<tr>
<td>NumLoc</td>
<td>Number of local searches done in preprocessing</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>The first local search begins with the user-specified starting point.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Subsequent local searches are done from judiciously chosen starting points.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If NumLoc is set to -1, local searches in preprocessing will be done until</td>
<td></td>
</tr>
<tr>
<td></td>
<td>proof of globality or MaxTime is reached. If NumLoc is set to -2, BARON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>decides the number of local searches in preprocessing based on problem and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NLP solver characteristics. NumLoc must be ≥ -2.</td>
<td></td>
</tr>
</tbody>
</table>

### 5.6 Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocRes</td>
<td>Option to control output from local search</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 no local search output</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 detailed results from local search will be printed to res.dat file</td>
<td></td>
</tr>
<tr>
<td>prfreq</td>
<td>Log output frequency in number of nodes</td>
<td>1000000</td>
</tr>
<tr>
<td>prlevel</td>
<td>Defines the level of log output printed.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 all log output is suppressed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 print log output</td>
<td></td>
</tr>
<tr>
<td>ptimefreq</td>
<td>Log output frequency in number of seconds</td>
<td>30</td>
</tr>
<tr>
<td>Results</td>
<td>Indicator if a results file is to be created</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 do not create file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 create file named according to the ResName option</td>
<td></td>
</tr>
</tbody>
</table>

### 5.7 Subsolver Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowExternal</td>
<td>Indicator for use of External NLP solver with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of external GAMS NLP solver as an NLP subsolver. 0 do not use the GAMS external NLP solver for local search 1 consider the GAMS external NLP solver for local search</td>
<td></td>
</tr>
<tr>
<td>AllowIpopt</td>
<td>Indicator for use of IPOPT with automatic NLP solver selection</td>
<td>1 (0 on MacOS X)</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of IPOPT as an NLP subsolver. 0 do not use IPOPT for local search 1 consider IPOPT for local search</td>
<td></td>
</tr>
<tr>
<td>AllowMinos</td>
<td>Indicator for use of MINOS with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of MINOS as an NLP subsolver. 0 do not use MINOS for local search 1 consider MINOS for local search</td>
<td></td>
</tr>
<tr>
<td>AllowSnopt</td>
<td>Indicator for use of SNOPT with automatic NLP solver selection</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of SNOPT as an NLP subsolver. 0 do not use SNOPT for local search 1 consider SNOPT for local search</td>
<td></td>
</tr>
<tr>
<td>ExtNLPsolver</td>
<td>External GAMS NLP solver and option file (e.g. conopt.1)</td>
<td>conopt</td>
</tr>
<tr>
<td></td>
<td>Specifies the GAMS NLP solver to be used when NLPSol is set to 6. All GAMS NLP solvers are available through this option. If a non-existing solver is specified or the solver chosen cannot solve NLPs, NLPSol will be reset to its default. A GAMS solver options file can be specified for the GAMS NLP solver by adding a dot followed by the options file number to the solver name, e.g., setting ExtNLPsolver to CONOPT.42 would instruct GAMS/CONOPT to use options file conopt.o42.</td>
<td></td>
</tr>
<tr>
<td>LPAlg</td>
<td>Specifies the LP algorithm to be used (available only with CPLEX as the LP solver)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 automatic selection of LP algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 primal simplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 dual simplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 barrier</td>
<td></td>
</tr>
<tr>
<td>LPSol</td>
<td>Specifies the LP/MIP Solver to be used</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3 CPLEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 XPRESS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 CLP</td>
<td></td>
</tr>
<tr>
<td>NLPSol</td>
<td>Specifies the NLP solver to be used</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>By default, BARON will select the NLP solver and may switch between different NLP solvers during the search, based on problem characteristics and solver performance. Any combination of licensed NLP solvers may be used in that case. A single specific NLP solver can be specified by setting this option to a value other than the default. If the specified solver is not licensed, BARON will default to automatic solver selection. -1 Automatic NLP solver selection and switching strategy 0 Local search based on function evaluations alone with no calls to local solvers 2 MINOS 4 SNOPT 6 GAMS NLP solver (see ExtNLPsolver) 9 IPOPT</td>
<td></td>
</tr>
</tbody>
</table>

### 5.8 Other Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>BilRel</td>
<td>Controls rounds of bilinear relaxations</td>
<td>4</td>
</tr>
<tr>
<td>CompIIS</td>
<td>Request the computation of an Irreducible Inconsistent Set (IIS)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>In case of an infeasible problem, this option can be used to search for an IIS. Setting this option 1, works very well for most problems. 0 do not search for an IIS 1 the search for an IIS is based on a fast heuristic 2 an IIS is obtained using a deletion filtering algorithm 3 an IIS is obtained using an addition filtering algorithm 4 an IIS is obtained using an addition-deletion filtering algorithm 5 an IIS is obtained using a depth-first search algorithm</td>
<td></td>
</tr>
<tr>
<td>CvxBt</td>
<td>Controls cvx-based tightening</td>
<td>1</td>
</tr>
<tr>
<td>CvxInitOA</td>
<td>Controls cvx-based initial outer approximation</td>
<td>1</td>
</tr>
<tr>
<td>CvxRel</td>
<td>Do cvxrel rounds of convexification cuts at every node</td>
<td>1</td>
</tr>
<tr>
<td>.EquClass</td>
<td>Equation Classification</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Specifies nature of constraint named EqName in the users model. Slices like supply.EquClass(&quot;new-york&quot;) 1 are allowed.</td>
<td></td>
</tr>
<tr>
<td>IISInt</td>
<td>Indicates whether general integers should be considered as potential members of the IIS When search for an IIS is requested through CompIIS, BARON assumes that the model is unlikely to include an error in terms of binaries, i.e., the binary definitions are assumed correct and the IIS output should be interpreted with respect to binary definitions. General integer bounds may be assumed as correct or can be questioned using the option IISInt. Integrality is enforced in both cases. 0 do not consider general integers as part of an IIS, assume them to be correct 1 consider general integers (but not binaries) as part of an IIS</td>
<td></td>
</tr>
<tr>
<td>InfBnd</td>
<td>infinity value to be used on bounds If set to 0, then no bounds are used.</td>
<td>0</td>
</tr>
<tr>
<td>MIPCuts</td>
<td>Do MIPCuts rounds of MIP cuts at every node</td>
<td>1</td>
</tr>
<tr>
<td>RLTRel</td>
<td>Do RLTRel rounds of RLT cuts at every node</td>
<td>1</td>
</tr>
</tbody>
</table>

Bibliography


CONOPT

Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

Contents

1 Introduction ................................................................. 58
2 Iteration Output ......................................................... 59
3 GAMS/CONOPT Termination Messages .................................. 60
4 Function Evaluation Errors ............................................... 64
5 The CONOPT Options File ................................................. 65
6 Hints on Good Model Formulation ....................................... 65
  6.1 Initial Values ......................................................... 65
  6.2 Bounds ............................................................... 66
  6.3 Simple Expressions .................................................. 67
  6.4 Scaling .............................................................. 68
7 NLP and DNLP Models ..................................................... 71
  7.1 DNLP Models: What Can Go Wrong? ................................. 71
  7.2 Reformulation from DNLP to NLP .................................. 72
  7.3 Smooth Approximations ............................................. 72
  7.4 Are DNLP Models Always Non-smooth? ............................... 73
  7.5 Are NLP Models Always Smooth? .................................. 74
8 APPENDIX A: Algorithmic Information ................................ 75
  8.1 Overview of GAMS/CONOPT ......................................... 75
  8.2 The CONOPT Algorithm ............................................ 75
  8.3 Iteration 0: The Initial Point ....................................... 76
  8.4 Iteration 1: Preprocessing .......................................... 77
  8.5 Iteration 2: Scaling .................................................. 83
  8.6 Finding a Feasible Solution: Phase 0 ................................ 84
  8.7 Finding a Feasible Solution: Phase 1 and 2 ................................ 85
  8.8 Linear and Nonlinear Mode: Phase 1 to 4 ............................ 85
  8.9 Linear Mode: The SLP Procedure .................................... 86
  8.10 Linear Mode: The Steepest Edge Procedure ......................... 87
  8.11 Nonlinear Mode: The SQP Procedure ................................ 87
  8.12 How to Select Non-default Options .................................. 88
  8.13 Miscellaneous Topics ................................................. 89
9 APPENDIX B - Options ..................................................... 95
  9.1 Algorithmic options ................................................ 95
  9.2 Debugging options .................................................. 96
  9.3 Output options ..................................................... 96
  9.4 Interface options .................................................... 97
10 APPENDIX C: References ................................................ 106
1 Introduction

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. Currently, there is a large number of different solvers available and the number is growing.

The most important distinction between the solvers is whether they attempt to find a local or a global solution. Solvers that attempt to find a global solution (so called Global Solvers) can usually not solve very large models. As a contrast most Local Solvers can work with much larger models, and models with over 10,000 variables and constraints are not unusual. If the model has the right mathematical properties, e.g. is convex, then Local Solvers will find a global optimum. Unfortunately, the mathematical machinery for testing whether a general NLP model is convex or not has not yet been developed (and is expected to be in the class or hard problems).

It is almost impossible to predict how difficult it is to solve a particular model with a particular algorithm, especially for NLP models, so GAMS cannot select the best algorithm for you automatically. When GAMS is installed you must select one of the nonlinear programming algorithms as the default solver for NLP models. If you want to switch between algorithms for a particular model you may add the statement "OPTION NLP = <solvername>;", in your GAMS source file before the $SOLVE statement, you may add "NLP = <solvername>", on the GAMS command line, or you may change the defaults using the GAMS IDE or by rerunning the GAMSINST program.

The only reliable way to find which solver to use for a particular class of models is so far to experiment. However, there are a few rules of thumb:

GAMS/CONOPT is well suited for models with very nonlinear constraints. If you experience that a solver has problems maintaining feasibility during the optimization you should try CONOPT. On the other hand, if you have a model with few nonlinearities outside the objective function then other solvers could be the best solver.

GAMS/CONOPT has a fast method for finding a first feasible solution that is particularly well suited for models with few degrees of freedom. If you have a model with roughly the same number of constraints as variable you should try CONOPT. CONOPT can also be used to solve square systems of equations without an objective function corresponding to the GAMS model class CNS - Constrained Nonlinear System.

GAMS/CONOPT can use second derivatives. If the number of variables is much larger than the number of constraints CONOPT will use second derivatives and overall progress can be considerably faster than for MINOS or SNOPT. IPOPT and KNITRO will also use second derivatives, but the method is very different and it is not possible to predict which solver will be better.

GAMS/CONOPT has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one then CONOPT will take advantage of this property. Similarly, intermediate variables only used to define objective terms are eliminated from the model and the constraints are moved into the objective function.

GAMS/CONOPT has many built-in tests and messages, and many models that can and should be improved by the modeler are rejected with a constructive message. CONOPT is therefore also a helpful debugging tool during model development. The best solver for the final, debugged model may or may not be CONOPT.

GAMS/CONOPT has been designed for large and sparse models. This means that both the number of variables and equations can be large. Indeed, NLP models with over 100,000 equations and variables have been solved successfully, and CNS models with over 1,000,000 equations and variables have also been solved. The components used to build CONOPT have been selected under the assumptions that the model is sparse, i.e. that most functions only depend on a small number of variables. CONOPT can also be used for denser models, but the performance will suffer significantly.

GAMS/CONOPT is designed for models with smooth functions, but it can also be applied to models that do not have differentiable functions, in GAMS called DNLP models. However, CONOPT will use the same algorithm used for a real NLP model and it will search for a point that satisfies standard first-order optimality conditions without taking into account that parts of the model could be non-smooth or non-differentiable. The lack of smoothness may confuse the algorithm in CONOPT causing slow convergence, and a point that satisfies standard first-order optimality conditions may not even exist. There are therefore no guarantees whatsoever for this class of models. If CONOPT terminates with a locally optimal solution then the solution will indeed be locally optimal. However, you will sometimes get termination messages like "Convergence too slow" or "No change in objective although the reduced gradient is greater than the tolerance" that indicate unsuccessful termination. The final point may or may not be locally optimal. If possible, you should try to reformulate a DNLP model to an equivalent or approximately equivalent form as described in section NLP and DNLP Models.
Most modelers should not be concerned with algorithmic details such as choice of algorithmic sub-components or tolerances. CONOPT has considerable build-in logic that selects a solution approach that seems to be best suited for the type of model at hand, and the approach is adjusted dynamically as information about the behavior of the model is collected and updated. The description of the CONOPT algorithm has therefore been moved to an appendix (Appendix A) and most modelers can skip it. However, if you are solving very large or complex models or if you are experiencing solution difficulties you may benefit from using non-standard tolerances or options, in which case you will need some understanding of what CONOPT is doing to your model. Some guidelines for selecting options can be found at the end of Appendix A and a list of all options and tolerances is shown in Appendix B.

The main text of this User’s Guide will give a short overview over the iteration output you will see on the screen (section Iteration Output), and explain the termination messages (section GAMS/CONOPT Termination Messages). We will then discuss function evaluation errors (section Function Evaluation Errors), the use of options (section The CONOPT Options File), and give a CONOPT perspective on good model formulation including topics such as initial values and bounds, simplification of expressions, and scaling (section Hints on Good Model Formulation). Finally, we will discuss the difference between NLP and DNLP models (section NLP and DNLP Models).

2 Iteration Output

On most machines you will by default get a logline on your screen or terminal at regular intervals. The iteration log may look something like this:

CONOPT 3 Jul 4, 2012 23.9.4 WEX 35892.35906 WEI x86_64/MS Windows

CONOPT 3 version 3.15G
Copyright (C) ARKI Consulting and Development A/S
Bagsvaerdvej 246 A
DK-2880 Bagsvaerd, Denmark

Iter Phase Ninf Infeasibility RGmax NSB Step InItr MX OK
0 0 1.6354151782E+01 (Input point) Pre-triangular equations: 2
Post-triangular equations: 1
1 0 1.5354151782E+01 (After pre-processing)
2 0 3.0983571843E+00 (After scaling)
10 0 12 3.0814290456E+00 0.0E+00 T T
20 0 12 3.0814290456E+00 0.0E+00 T T
30 0 13 3.0814290456E+00 0.0E+00 F F
40 0 18 2.3738740159E+00 2.3E-02 T T
50 0 23 2.1776589484E+00 0.0E+00 F F

Iter Phase Ninf Infeasibility RGmax NSB Step InItr MX OK
60 0 33 2.1776589484E+00 0.0E+00 T T
70 0 43 2.1776589484E+00 0.0E+00 F F
80 0 53 2.1776589484E+00 0.0E+00 F F
90 0 63 2.1776589484E+00 0.0E+00 F F
100 0 73 2.1776589484E+00 0.0E+00 F F
110 0 83 2.1776589484E+00 0.0E+00 F F
120 0 93 2.1776589484E+00 0.0E+00 F F
130 0 103 2.1776589484E+00 0.0E+00 F F
140 0 113 2.1776589484E+00 0.0E+00 T T
150 0 119 8.7534351971E-01 0.0E+00 F F
The first few lines identify the version of CONOPT you use.

The first iterations have a special interpretation: iteration 0 represents the initial point exactly as received from GAMS, iteration 1 represent the point that is the result of CONOPT’s pre-processing, and iteration 2 represents the same point after scaling (even if scaling is turned off).

The remaining iterations are characterized by the value of "Phase" in column 2. The model is infeasible during Phase 0, 1, and 2 and the Sum of Infeasibilities in column 4 (labeled "Infeasibility") is being minimized; the model is feasible during Phase 3 and 4 and the actual objective function, also shown in column 4 (now labeled "Objective"), is minimized or maximized. Phase 0 iterations are Newton-like iterations. They are very cheap so you should not be concerned if there are many of these Phase 0 iterations. During Phase 1 and 3 the model behaves almost linearly and CONOPT applies special linear iterations that take advantage of the linearity. These iterations are sometimes augmented with some inner "Sequential Linear Programming" (SLP) iterations, indicated by a number of inner SLP iterations in the "InItr" column. During Phase 2 and 4 the model behaves more nonlinear and most aspects of the iterations are therefore changed: the line search is more elaborate, and CONOPT needs second order information to improve the convergence. For small and simple models CONOPT will approximate second order information as a byproduct of the line searches. For larger and more complex models CONOPT will use some inner "Sequential Quadratic Programming" (SQP) iterations based on exact second derivatives. These SQP iterations are identified by the number of inner SQP iterations in the "InItr" column.

The column "NSB" for Number of SuperBasics defines the degree of freedom or the dimension of the current search space, and "Rgmax" measures the largest reduced gradient among the non-optimal variables. Rgmax should eventually converge towards zero, but convergence is not expected to be monotone. The last two columns labeled "MX" and "OK" gives information about the line search: OK = T means that the line search was well-behaved, and OK = F means that the line search was terminated before an optimal step length was found because it was not possible to find a feasible solution for large step lengths. MX = T means that the line search was terminated by a variable reaching a bound (always combined with OK = T), and MX = F means that the step length was determined by nonlinearities. If OK = T then the step length was optimal; if OK = F then the constraints were too nonlinear to allow CONOPT to make a full optimal step.

3 GAMS/CONOPT Termination Messages

GAMS/CONOPT may terminate in a number of ways. This section will show most of the termination messages and explain their meaning. It will also show the Model Status returned to GAMS in "<model>.Modelstat", where "model>" represents the name of the GAMS model. The Solver Status returned in "<model>.Solvestat" will be given if it is different from 1 (Normal Completion). We will in all cases first show the message from CONOPT followed by a short explanation. The first 4
messages are used for optimal solutions and CONOPT will return Modelstat = 2 (Locally Optimal), except as noted below:

** Optimal solution. There are no superbasic variables.

The solution is a locally optimal corner solution. The solution is determined by constraints only, and it is usually very accurate. In some cases CONOPT can determine that the solution is globally optimal and it will return Modelstat = 1 (Optimal).

** Optimal solution. Reduced gradient less than tolerance.

The solution is a locally optimal interior solution. The largest component of the reduced gradient is less than the tolerance \( rtredg \) with default value 1.e-7. The value of the objective function is very accurate while the values of the variables are less accurate due to a flat objective function in the interior of the feasible area.

** Optimal solution. The error on the optimal objective function value estimated from the reduced gradient and the estimated Hessian is less than the minimal tolerance on the objective.

The solution is a locally optimal interior solution. The largest component of the reduced gradient is larger than the tolerance \( rtredg \). However, when the reduced gradient is scaled with information from the estimated Hessian of the reduced objective function the solution seems optimal. For this to happen the objective must be large or the reduced objective must have large second derivatives so it is advisable to scale the model. See the sections on "Scaling" and "Using the Scale Option in GAMS" for details on how to scale a model.

** Optimal solution. Convergence too slow. The change in objective has been less than xx.xx for xx consecutive iterations.

CONOPT stops with a solution that seems optimal. The solution process is stopped because of slow progress. The largest component of the reduced gradient is greater than the optimality tolerance \( rtredg \), but less than \( rtredg \) multiplied by the largest Jacobian element divided by 100. The model must have large derivatives so it is advisable to scale it.

The four messages above all exist in versions where "Optimal" is replaced by "Infeasible" and Modelstat will be 5 (Locally Infeasible) or 4 (Infeasible). The infeasible messages indicate that a Sum of Infeasibility objective function is locally minimal, but positive. If the model is convex it does not have a feasible solution; if the model is non-convex it may have a feasible solution in a different region. See the section on "Initial Values" for hints on what to do.

** Feasible solution. Convergence too slow. The change in objective has been less than xx.xx for xx consecutive iterations.

** Feasible solution. The tolerances are minimal and there is no change in objective although the reduced gradient is greater than the tolerance.

The two messages above tell that CONOPT stops with a feasible solution. In the first case the solution process is very slow and in the second there is no progress at all. However, the optimality criteria have not been satisfied. These messages are accompanied by Modelstat = 7 (Feasible Solution) and Solvestat = 4 (Terminated by Solver). The problem can be caused by discontinuities if the model is of type DNLP; in this case you should consider alternative, smooth formulations as discussed in section NLP and DNLP Models . The problem can also be caused by a poorly scaled model. See section Scaling for hints on model scaling. Finally, it can be caused by stalling as described in section Stalling in Appendix A. The two messages also exist in a version where "Feasible" is replaced by "Infeasible". Modelstat is in this case 6 (Intermediate Infeasible) and Solvestat is still 4 (Terminated by Solver); these versions tell that CONOPT cannot make progress towards feasibility, but the Sum of Infeasibility objective function does not have a well defined local minimum.
**Unbounded solution. A variable has reached 'infinity'.**

Largest legal value ($rtmaxv$) is $xx.xx$

CONOPT considers a solution to be unbounded if a variable exceeds the indicated value of $rtmaxv$ (default 1.e10) and it returns with Modelstat = 3 (Unbounded). The check for unboundedness is done at every iteration which means that CONOPT will stop if an intermediate solution has a variable that is very large, even if none of the variables in the optimal solution have large values. Check whether the solution appears unbounded or the problem is caused by the scaling of the unbounded variable $<\text{var}>$ mentioned in the first line of the message. If the model seems correct you are advised to scale it. There is also a lazy solution: you can increase the largest legal value, $rtmaxv$, as mentioned in the section on options. However, you will pay through reduced reliability or increased solution times. Unlike LP models, where an unbounded model is recognized by an unbounded ray and the iterations are stopped far from “infinity”, CONOPT will actually have to make a line search and move to a region with large values of the variables. This may lead to bad scaling and to many different kinds of tolerance and roundoff problems, including problems of determining whether a solution is feasible or not.

The message above exists in a version where “Unbounded” is replaced by ”Infeasible” and Modelstat is 5 (Locally Infeasible). You may also see a message like

**Free variable becomes too large**

*Infeasible solution. A free variable exceeds the allowable range. Current value is 1.02E+10 and current upper bound ($rtmaxv$) is 1.00E+10*

These two messages indicate that some variables become very large before a feasible solution has been found. You should again check whether the problem is caused by the scaling of the unbounded variable $<\text{var}>$ mentioned in the first line of the message. If the model seems correct you should scale it.

**The time limit has been reached.**

The time or resource limit defined in GAMS, either by default (usually 1000 seconds) or by "OPTION RESLIM = xx;" or "<model>.RESLIM = xx;" statements, has been reached. CONOPT will return with Solvestat = 3 (Resource Interrupt) and Modelstat either 6 (Locally Infeasible) or 7 (Feasible Solution).

**The iteration limit has been reached.**

The iteration limit defined in GAMS, either by default (usually 2000000000 iterations) or by "OPTION ITERLIM = xx;" or "<model>.ITERLIM = xx;" statements, has been reached. CONOPT will return with Solvestat = 2 (Iteration Interrupt) and Modelstat either 6 (Locally Infeasible) or 7 (Feasible Solution).

**Domain errors in nonlinear functions.**

Check bounds on variables.

The number of function evaluation errors has reached the limit defined in GAMS by "OPTION DOMLIM = xx;" or "<model>.DOMLIM = xx;" statements or the default limit of 0 function evaluation errors. CONOPT will return with Solvestat = 5 (Evaluation Error Limit) and Modelstat either 6 (Locally Infeasible) or 7 (Feasible Solution). See more details in section Function Evaluation Errors.

**An initial derivative is too large (larger than $rtmaxj= xx.xx$)**

Scale the variables and/or equations or add bounds.

$<\text{var}>$ appearing in

$<\text{equ}>$: Initial Jacobian element too large = $xx.xx$

and
** A derivative is too large (larger than \(Rtmaxj = xx.xx\)).
Scale the variables and/or equations or add bounds.

\(<\text{var}>\) appearing in
\(<\text{equ}>\): Jacobian element too large = xx.xx

These two messages appear if a derivative or Jacobian element is very large, either in the initial point or in a later intermediate point. The relevant variable and equation pair(s) will show you where to look. A large derivative means that the function changes very rapidly even after a very small change in the variable and it will most likely create numerical problems for many parts of the optimization algorithm. Instead of attempting to solve a model that most likely will fail, CONOPT will stop and you are advised to adjust the model if at all possible.

If the offending derivative is associated with a \(\log(X)\) or \(1/X\) term you may try to increase the lower bound on \(X\). If the offending derivative is associated with an \(\exp(X)\) term you must decrease the upper bound on \(X\). You may also try to scale the model, either manually or using the variable.SCALE and/or equation.SCALE option in GAMS as described in section Scaling. There is also in this case a lazy solution: increase the limit on Jacobian elements, \(rtmaxj\); however, you will pay through reduced reliability or longer solution times.

In addition to the messages shown above you may see messages like

** An equation in the pre-triangular part of the model cannot be solved because the critical variable is at a bound.

** An equation in the pre-triangular part of the model cannot be solved because of too small pivot.

or

** An equation is inconsistent with other equations in the pre-triangular part of the model.

These messages containing the word "Pre-triangular" are all related to infeasibilities identified by CONOPT's pre-processing stage and they are explained in detail in section Iteration 1: Preprocessing in Appendix A.

Usually, CONOPT will be able to estimate the amount of memory needed for the model based on statistics provided by GAMS. However, in some cases with unusual models, e.g. very dense models or very large models, the estimate will be too small and you must request more memory yourself using a statement like "\(<\text{model}>\).WORKFACTOR = x.x;" or "\(<\text{model}>\).WORKSPACE = xx;" in GAMS or by adding "workfactor=xx" to the command line call of GAMS. The message you will see is similar to the following:

** FATAL ERROR ** Insufficient memory to continue the optimization.

You must request more memory.

Current CONOPT space = 0.29 Mbytes
Estimated CONOPT space = 0.64 Mbytes
Minimum CONOPT space = 0.33 Mbytes

CONOPT time Total 0.109 seconds
of which: Function evaluations 0.000 = 0.0%
1st derivative evaluations 0.000 = 0.0%

The text after "Insufficient memory to" may be different; it says something about where CONOPT ran out of memory. If the memory problem appears during model setup the message will be accompanied by Solvestat = 13 (System Failure) and Modelstat = 13 (Error No Solution) and CONOPT will not return any values. If the memory problem appears later during the optimization Solvestat will be 11 (Internal Solver Failure) and Modelstat will be either 6 (Intermediate Infeasible) or 7
(Feasible Solution) and CONOPT will return primal solution values. The marginals of both equations and variables will be zero or EPS.

It is recommended that you use the WORKFACTOR option if you must change the amount of memory. The same number will usually work for a whole family of models. If you prefer to use WORKSPACE, the GAMS WORKSPACE option corresponds to the amount of memory, measured in Mbytes.

A new termination message has been added from version 3.16C:

** Feasible solution. The solution process has been terminated because intermediate results have become NaN (Not A Number).**

and similar with Infeasible. To prevent non-sensible results and/or infinite loops in special degenerate cases CONOPT has added checks for internal intermediate results being NaN (Not A Number) or very large. If this happens CONOPT will try to change some tolerances and try to continue the optimization. If this attempt fails CONOPT will stop and return the message above. The solver status will return 4 "Terminated by Solver" and model status 6 or 7, "Intermediate Infeasible" or "Intermediate Feasible." Section Overflow and NaN (Not A Number) in Appendix A has more details on the sources of NaN and the actions that can be taken by the user and by CONOPT.

4 Function Evaluation Errors

Many of the nonlinear functions available with GAMS are not defined for all values of their arguments. LOG is not defined for negative arguments, EXP overflows for large arguments, and division by zero is illegal. To avoid evaluating functions outside their domain of definition you should add reasonable variable bounds. CONOPT will in return guarantee that the nonlinear functions never are evaluated with variables outside their bounds.

In some cases bounds are not sufficient, e.g. in the expression LOG( SUM(I, X(I) ) ): in some models each individual X should be allowed to become zero, but the SUM should not. In this case you should introduce an intermediate variable and an extra equation, e.g. XSUMDEF .. XSUM =E= SUM(I,X(I)); add a lower bound on XSUM; and use XSUM as the argument to the LOG function. See section Simple Expressions for additional comments on this topic.

Whenever a nonlinear function is called outside its domain of definition, GAMS' function evaluator will intercept the function evaluation error and prevent the system to crash. GAMS will replace the undefined result by some appropriate real number, and it will make sure the error is reported to the modeler as part of the standard solution output in the GAMS listing file. GAMS will also report the error to CONOPT, so CONOPT can try to correct the problem by backtracking to a safe point. Finally, CONOPT will be instructed to stop after DOMLIM errors.

During Phase 0, 1, and 3 CONOPT will often use large steps as the initial step in a line search and functions will very likely be called with some of the variables at their lower or upper bound. You are therefore likely to get a division-by-zero error if your model contains a division by X and X has a lower bound of zero. And you are likely to get an exponentiation overflow error if your model contains EXP(X) and X has no upper bound. However, CONOPT will usually not get trapped in a point outside the domain of definition for the model. When GAMS' function evaluator reports that a point is "bad", CONOPT will decrease the step length, and it will for most models be able to recover and continue to an optimal solution. It is therefore safe to use a large value for DOMLIM instead of GAMS default value of 0.

CONOPT may get stuck in some cases, for example because there is no previous point to backtrack to, because "bad" points are very close to "reasonable" feasible points, or because the derivatives are not defined in a feasible point. The more common messages are:

** Fatal Error ** Function error in initial point in Phase 0 procedure.

** Fatal Error ** Function error after small step in Phase 0 procedure.

** Fatal Error ** Function error very close to a feasible point.

** Fatal Error ** Function error while reducing tolerances.
** Fatal Error ** Function error in Pre-triangular equations.

** Fatal Error ** Function error after solving Pre-triangular equations.

** Fatal Error ** Function error in Post-triangular equation.

In the first four cases you must either add better bounds or define better initial values. If the problem is related to a pre- or post-triangular equation as shown by the last three messages then you can turn part of the pre-processing off as described in section Iteration 1: Preprocessing in Appendix A. However, this may make the model harder to solve, so it is usually better to add bounds and/or initial values.

5 The CONOPT Options File

CONOPT has been designed to be self-tuning. Most tolerances are dynamic. As an example: The feasibility of a constraint is always judged relative to the dual variable on the constraint and relative to the expected change in objective in the coming iteration. If the dual variable is large then the constraint must be satisfied with a small tolerance, and if the dual variable is small then the tolerance is larger. When the expected change in objective in the first iterations is large then the feasibility tolerances are also large. And when we approach the optimum and the expected change in objective becomes smaller then the feasibility tolerances become smaller.

Because of the self-tuning nature of CONOPT you should in most cases be well off with default tolerances. If you do need to change some tolerances, possibly following the advice in Appendix A, it can be done in the CONOPT Options file. The name of the CONOPT Options file is on most systems "conopt.opt". You must tell the solver that you want to use an options file with the statement "<model>.OPTFILE = 1" in your GAMS source file before the SOLVE statement or with "optfile = 1" on the command line.

The format of the CONOPT Options file consists in its simplest form of a number of lines like these:

```
rtmaxv = 1.e12
lfnsup = 500
```

Upper case letters are converted to lower case so the second line could also be written as "LFSUP = 500". The value must be written using legal GAMS format, i.e. a real number may contain an optional E exponent, but a number may not contain blanks. The value must have the same type as the option, i.e. real options must be assigned real values, integer options must be assigned integer values, and logical options must be assigned logical values. The logical value representing true are true, t, yes, or 1, and the logical values representing false are false, f, no, or 0.

In previous versions of CONOPT you could add "SET" in front of the option assignment. This is no longer supported. You can still replace the equal sign with := and you can add end of line comments after a # or ! character. Lines starting with * in column 1 are treated as comment lines.

6 Hints on Good Model Formulation

This section will contain some comments on how to formulate a nonlinear model so it becomes easier to solve with CONOPT. Most of the recommendations will be useful for any nonlinear solver, but not all. We will try to mention when a recommendation is CONOPT specific.

6.1 Initial Values

Good initial values are important for many reasons. Initial values that satisfy or closely satisfy many of the constraints reduces the work involved in finding a first feasible solution. Initial values that in addition are close to the optimal ones also reduce the distance to the final point and therefore indirectly the computational effort. The progress of the optimization
algorithm is based on good directional information and therefore on good derivatives. The derivatives in a nonlinear model depend on the current point, and the initial point in which the initial derivatives are computed is therefore again important. Finally, non-convex models may have multiple solutions, but the modeler is looking for one in a particular part of the search space; an initial point in the right neighborhood is more likely to return the desired solution.

The initial values used by CONOPT are all coming from GAMS. The initial values used by GAMS are by default the value zero projected on the bounds. I.e. if a variable is free or has a lower bound of zero, then its default initial value is zero. Unfortunately, zero is in many cases a bad initial value for a nonlinear variable. An initial value of zero is especially bad if the variable appears in a product term since the initial derivative becomes zero, and it appears as if the function does not depend on the variable. CONOPT will warn you and ask you to supply better initial values if the number of derivatives equal to zero is larger than 20 percent.

If a variable has a small positive lower bound, for example because it appears as an argument to the LOG function or as a denominator, then the default initial value is this small lower bound and it is also bad since this point will have very large first and second derivatives.

You should therefore supply as many sensible initial values as possible by making assignment to the level value, var.L, in GAMS. An easy possibility is to initialize all variables to 1, or to the scale factor if you use GAMS’ scaling option. A better possibility is to select reasonable values for some variables that from the context are known to be important, and then use some of the equations of the model to derive values for other variables. A model may contain the following equation:

$$PMDEF(IT) \ldots PM(IT) = E= PWM(IT) \times ER \times (1 + TM(IT)) ;$$

where PM, PWM, and ER are variables and TM is a parameter. The following assignment statements use the equation to derive consistent initial values for PM from sensible initial values for PWM and ER:

$$ER.L = 1; PWM.L(IT) = 1; PM.L(IT) = PWM.L(IT) \times ER.L \times (1 + TM(IT)) ;$$

With these assignments equation PMDEF will be feasible in the initial point, and since CONOPT uses a feasible path method it will remain feasible throughout the optimization (unless the pre-processor destroys it, see section Iteration 1: Preprocessing in Appendix A).

If CONOPT has difficulties finding a feasible solution for your model you should try to use this technique to create an initial point in which as many equations as possible are satisfied. You may also try the optional Crash procedure described in section Preprocessing: The Optional Crash Procedure in Appendix A by adding the line ”lstcrs=t” to the CONOPT options file. The crash procedure tries to identify equations with a mixture of un-initialized variables and variables with initial values, and it solves the equations with respect to the un-initialized variables; the effect is similar to the manual procedure shown above.

### 6.2 Bounds

Bounds have two purposes in nonlinear models. Some bounds represent constraints on the reality that is being modeled, e.g. a variable must be positive. These bounds are called model bounds. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called algorithmic bounds.

Model bounds have natural roots and do not cause any problems. Algorithmic bounds require a closer look at the functional form of the model. The content of a LOG should be greater than say 1.e-3, the content of an EXP should be less than 5 to 8, and a denominator should be greater than say 1.e-2. These recommended lower bounds of 1.e-3 and 1.e-2 may appear to be unreasonably large. However, both LOG(X) and 1/X are extremely nonlinear for small arguments. The first and second derivatives of LOG(X) at X=1.e-3 are 1.e+3 and -1.e6, respectively, and the first and second derivatives of 1/X at X=1.e-2 are -1.e+4 and 2.e+6, respectively.

If the content of a LOG or EXP function or a denominator is an expression then it may be advantageous to introduce a bounded intermediate variable as discussed in the next section.

Note that bounds in some cases can slow the solution process down. Too many bounds may for example introduce degeneracy. If you have constraints of the following type

$$VUB(I) \ldots X(I) \leq Y;$$
or

\[
Y_{SUM} \quad \text{..} \quad Y = E= \text{SUM( I, X(I) )};
\]

and X is a POSITIVE VARIABLE then you should in general not declare Y a POSITIVE VARIABLE or add a lower bound of zero on Y. If Y appears in a nonlinear function you may need a strictly positive bound. Otherwise, you should declare Y a free variable; CONOPT will then make Y basic in the initial point and Y will remain basic throughout the optimization. New logic in CONOPT tries to remove this problem by detecting when a harmful bound is redundant so it can be removed, but it is not yet a fool proof procedure.

Section Iteration 1: Preprocessing in Appendix A gives another example of bounds that can be counter productive.

### 6.3 Simple Expressions

The following model component

\[
\begin{align*}
\text{PARAMETER} & \quad \text{MU(I)}; \\
\text{VARIABLE} & \quad X(I), S(I), OBJ; \\
\text{EQUATION} & \quad \text{OBJDEF}; \\
\text{OBJDEF} \quad \text{..} & \quad \text{OBJ} = E= \exp( \text{SUM( I, SQR( X(I) - MU(I) ) / S(I) )} );
\end{align*}
\]

can be re-written in the slightly longer but simpler form

\[
\begin{align*}
\text{PARAMETER} & \quad \text{MU(I)}; \\
\text{VARIABLE} & \quad X(I), S(I), OBJ, INTERM; \\
\text{EQUATION} & \quad \text{INTDEF}, \text{OBJDEF}; \\
\text{INTDEF} \quad \text{..} & \quad \text{INTERM} = E= \text{SUM( I, SQR( X(I) - MU(I) ) / S(I) )}; \\
\text{OBJDEF} \quad \text{..} & \quad \text{OBJ} = E= \exp( \text{INTERM} );
\end{align*}
\]

The first formulation has very complex derivatives because EXP is taken of a long expression. The second formulation has much simpler derivatives; EXP is taken of a single variable, and the variables in INTDEF appear in a sum of simple independent terms.

In general, try to avoid nonlinear functions of expressions, divisions by expressions, and products of expressions, especially if the expressions depend on many variables. Define intermediate variables that are equal to the expressions and apply the nonlinear function, division, or product to the intermediate variable. The model will become larger, but the increased size is taken care of by CONOPT's sparse matrix routines, and it is compensated by the reduced complexity. If the model is solved with CONOPT using explicit second derivatives then simple expressions will result in sparser second derivatives that are both faster to compute and to use.

The reduction in complexity can be significant if an intermediate expression is linear. The following model fragment:

\[
\begin{align*}
\text{VARIABLE} & \quad X(I), Y; \\
\text{EQUATION} & \quad \text{YDEF}; \\
\text{YDEF} \quad \text{..} & \quad Y = E= 1 / \text{SUM(I, X(I) )};
\end{align*}
\]

should be written as

\[
\begin{align*}
\text{VARIABLE} & \quad X(I), XSUM, Y; \\
\text{EQUATION} & \quad \text{XSUMDEF}, \text{YDEF}; \\
\text{XSUMDEF} \quad \text{..} & \quad XSUM = E= \text{SUM(I, X(I) )}; \\
\text{YDEF} \quad \text{..} & \quad Y = E= 1 / XSUM; \\
\text{XSUM.LO} & = 1.E-2;
\end{align*}
\]

for three reasons. First, because the number of nonlinear derivatives is reduced in number and complexity. Second, because the lower bound on the intermediate result will bound the search away from the singularity at XSUM = 0. And third, because the matrix of second derivatives for the last model only depend on XSUM while it depends on all X in the first model.
The last example shows an added potential saving by expanding functions of linear expressions. A constraint depends in a nonlinear fashion on the accumulated investments, INV, like

\[ \text{CON}(I) \ .. \ f \left( \text{SUM} \left( J \$(\text{ORD}(J) \leq \text{ORD}(I)), \text{INV}(J) \right) \right) =L= B(I); \]

A new intermediate variable, \( \text{CAP}(I) \), that is equal to the content of the SUM can be defined recursively with the constraints

\[ \text{CDEF}(I) \ .. \ \text{CAP}(I) =E= \text{INV}(I) + \text{CAP}(I-1); \]

and the original constraints become

\[ \text{CON}(I) \ .. \ f( \text{CAP}(I) ) =L= B(I); \]

The reformulated model has \( N \) additional variables and \( N \) additional linear constraints. In return, the original \( N \) complex nonlinear constraints have been changed into \( N \) simpler nonlinear constraints. And the number of Jacobian elements, that has a direct influence on much of the computational work both in GAMS and in CONOPT, has been reduced from \( N \times (N+1)/2 \) nonlinear elements to \( 3 \times N - 1 \) linear elements and only \( N \) nonlinear element. If \( f \) is an invertible increasing function you may even rewrite the last constraint as a simple bound:

\[ \text{CAP}.\text{LO}(I) = \text{finv}(B(I)); \]

Some NLP solvers encourage you to move as many nonlinearities as possible into the objective which may make the objective very complex. This is neither recommended nor necessary with CONOPT. A special pre-processing step (discussed in section Iteration 1: Preprocessing in Appendix A) will aggregate parts of the model if it is useful for CONOPT without increasing the complexity in GAMS.

6.4 Scaling

Nonlinear as well as Linear Programming Algorithms use the derivatives of the objective function and the constraints to determine good search directions, and they use function values to determine if constraints are satisfied or not. The scaling of the variables and constraints, i.e. the units of measurement used for the variables and constraints, determine the relative size of the derivatives and of the function values and thereby also the search path taken by the algorithm.

Assume for example that two goods of equal importance both cost $1 per kg. The first is measured in gram, the second in tons. The coefficients in the cost function will be $1000/g and $0.001/ton, respectively. If cost is measured in $1000 units then the coefficients will be 1 and 1.e-6, and the smaller may be ignored by the algorithm since it is comparable to some of the zero tolerances.

CONOPT assumes implicitly that the model to be solved is well scaled. In this context well scaled means:

- Basic and superbasic solution values are expected to be around 1, e.g. from 0.01 to 100. Nonbasic variables will be at a bound, and the bound values should not be larger than say 100.
- Dual variables (or marginals) on active constraints are expected to be around 1, e.g. from 0.01 to 100. Dual variables on non-binding constraints will of course be zero.
- Derivatives (or Jacobian elements) are expected to be around 1, e.g. from 0.01 to 100.

Variables become well scaled if they are measured in appropriate units. In most cases you should select the unit of measurement for the variables so their expected value is around unity. Of course there will always be some variation. Assume \( X(I) \) is the production at location \( I \). In most cases you should select the same unit of measurement for all components of \( X \), for example a value around the average capacity.

Equations become well scaled if the individual terms are measured in appropriate units. After you have selected units for the variables you should select the unit of measurement for the equations so the expected values of the individual terms are around one. If you follow these rules, material balance equations will usually have coefficients of plus and minus one.
Derivatives will usually be well scaled whenever the variables and equations are well scaled. To see if the derivatives are well scaled, run your model with a positive OPTION LIMROW and look for very large or very small coefficients in the equation listing in the GAMS output file.

CONOPT computes a measure of the scaling of the Jacobian, both in the initial and in the final point, and if it seems large it will be printed. The message looks like:

** WARNING ** The variance of the derivatives in the initial point is large (= 4.1 ). A better initial point, a better scaling, or better bounds on the variables will probably help the optimization.

The variance is computed as SQRT(SUM(LOG(ABS(Jac(i)))**2)/NZ) where Jac(i) represents the NZ nonzero derivatives (Jacobian elements) in the model. A variance of 4.1 corresponds to an average value of LOG(JAC)**2 of 4.1**2, which means that Jacobian values outside the range EXP(-4.1)=0.017 to EXP(+4.1)=60.4 are about as common at values inside. This range is for most models acceptable, while a variance of 5, corresponding to about half the derivatives outside the range EXP(-5)=0.0067 to EXP(+5)=148, can be dangerous.

### Scaling of Intermediate Variables

Many models have a set of variables with a real economic or physical interpretation plus a set of intermediate or helping variables that are used to simplify the model. We have seen some of these in section Simple Expressions on Simple Expressions. It is usually rather easy to select good scaling units for the real variables since we know their order of magnitude from economic or physical considerations. However, the intermediate variables and their defining equations should preferably also be well scaled, even if they do not have an immediate interpretation. Consider the following model fragment where X, Y, and Z are variables and Y is the intermediate variable:

```gams
SET P / P0*P4 /
PARAMETER A(P) / P0 211, P1 103, P2 42, P3 31, P4 6 /
YDEF .. Y =E= SUM(P, A(P)*POWER(X,ORD(P)-1));
ZDEF .. Z =E= LOG(Y);
```

X lies in the interval 1 to 10 which means that Y will be between 211 and 96441 and Z will be between 5.35 and 11.47. Both X and Z are reasonably scaled while Y and the terms and derivatives in YDEF are about a factor 1.e4 too large. Scaling Y by 1.e4 and renaming it YS gives the following scaled version of the model fragment:

```gams
YDEFS1 .. YS =E= SUM(P, A(P)*POWER(X,ORD(P)-1))*1.E-4;
ZDEFS1 .. Z =E= LOG(YS*1.E4);
```

The Z equation can also be written as

```gams
ZDEFS2 .. Z =E= LOG(YS) + LOG(1.E4);
```

Note that the scale factor 1.e-4 in the YDEFS1 equation has been placed on the right hand side. The mathematically equivalent equation

```gams
YDEFS2 .. YS*1.E4 =E= SUM(P, A(P)*POWER(X,ORD(P)-1));
```

will give a well scaled YS, but the right hand side terms of the equation and their derivatives have not changed from the original equation YDEF and they are still far too large.

### Using the Scale Option in GAMS

The rules for good scaling mentioned above are exclusively based on algorithmic needs. GAMS has been developed to improve the effectiveness of modelers, and one of the best ways seems to be to encourage modelers to write their models
using a notation that is as "natural" as possible. The units of measurement is one part of this natural notation, and there is unfortunately often a conflict between what the modeler thinks is a good unit and what constitutes a well scaled model.

To facilitate the translation between a natural model and a well scaled model GAMS has introduced the concept of a scale factor, both for variables and equations. The notation and the definitions are quite simple. First of all, scaling is by default turned off. To turn it on, enter the statement "<model>.SCALEOPT = 1;" in your GAMS program somewhere after the MODEL statement and before the SOLVE statement. "<model>" is the name of the model to be solved. If you want to turn scaling off again, enter the statement "<model>.SCALEOPT = 0;" somewhere before the next SOLVE.

The scale factor of a variable or an equation is referenced with the suffix ".SCALE", i.e. the scale factor of variable X(I) is referenced as X.SCALE(I). Note that there is one scale value for each individual component of a multidimensional variable or equation. Scale factors can be defined in assignment statements with X.SCALE(I) on the left hand side, and scale factors, both from variables and equations, can be used on the right hand side, for example to define other scale factors. The default scale factor is always 1, and a scale factor must be positive; GAMS will generate an execution time error if the scale factor is less than 1.e-20.

The mathematical definition of scale factors is as follows: The scale factor on a variable, \( V^s \), is used to relate the variable as seen by the modeler, \( V^m \), to the variable as seen by the algorithm, \( V^a \), as follows:

\[
V^m = V^a \times V^s
\]

This means, that if the variable scale, \( V^s \), is chosen to represent the order of magnitude of the modeler’s variable, \( V^m \), then the variable seen by the algorithm, \( V^a \), will be around 1. The scale factor on an equation, \( G^s \), is used to relate the equation as seen by the modeler, \( G^m \), to the equation as seen by the algorithm, \( G^a \), as follows:

\[
G^m = G^a \times G^s
\]

This means, that if the equation scale, \( G^s \), is chosen to represent the order of magnitude of the individual terms in the modelers version of the equation, \( G^m \), then the terms seen by the algorithm, \( G^a \), will be around 1.

The derivatives in the scaled model seen by the algorithm, i.e. \( dG^a/dV^a \), are related to the derivatives in the modelers model, \( dG^m/dV^m \), through the formula:

\[
dG^a/dV^a = dG^m/dV^m \times V^s/G^s
\]

i.e. the modelers derivative is multiplied by the scale factor of the variable and divided by the scale factor of the equation. Note, that the derivative is unchanged if \( V^s = G^s \). Therefore, if you have a GAMS equation like

\[ \text{G .. V =E= expression; } \]

and you select \( G^s = V^s \) then the derivative of V will remain 1. If we apply these rules to the example above with an intermediate variable we can get the following automatic scale calculation, based on an "average" reference value for \( X \):

```gams
SCALAR XREF; XREF = 6;
Y.SCALE = SUM(P, A(P)*POWER(XREF,ORD(P)-1));
YDEF.SCALE = Y.SCALE;
```

or we could scale \( Y \) using values at the end of the \( X \) interval and add safeguards as follows:

```gams
Y.SCALE = MAX( ABS(SUM(P, A(P)*POWER(X.LO,ORD(P)-1))),
                  ABS(SUM(P, A(P)*POWER(X.UP,ORD(P)-1))),
                  0.01 );
```

Lower and upper bounds on variables are automatically scaled in the same way as the variable itself. Integer and binary variables cannot be scaled.

GAMS’ scaling is in most respects hidden for the modeler. The solution values reported back from a solution algorithm, both primal and dual, are always reported in the user’s notation. The algorithm’s versions of the equations and variables are
only reflected in the derivatives in the equation and column listings in the GAMS output if `OPTION LIMROW` and/or `LIMCOL` are positive, and in debugging output from the solution algorithm, generated with `OPTION SYSOUT = ON`. In addition, the numbers in the algorithms iteration log will represent the scaled model: the infeasibilities and reduced gradients will correspond to the scaled model, and if the objective variable is scaled, the value of the objective function will be the scaled value.

A final warning about scaling of multidimensional variables is appropriate. Assume variable $X(I,J,K)$ only appears in the model when the parameter $IJK(I,J,K)$ is nonzero, and assume that $\text{CARD}(I) = \text{CARD}(J) = \text{CARD}(K) = 100$ while $\text{CARD}(IJK)$ is much smaller than $100^{**2} = 1.e6$. Then you should only scale the variables that appear in the model, i.e.

$$X.SCALE(I,J,K)\$IJK(I,J,K) = \text{expression};$$

The statement

$$X.SCALE(I,J,K) = \text{expression};$$

will generate records for $X$ in the GAMS database for all combinations of $I$, $J$, and $K$ for which the expression is different from 1, i.e. up to $1.e6$ records, and apart from spending a lot of time you will very likely run out of memory. Note that this warning also applies to non-default lower and upper bounds.

### 7 NLP and DNLP Models

GAMS has two classes of nonlinear model, NLP and DNLP. NLP models are defined as models in which all functions that appear with endogenous arguments, i.e. arguments that depend on model variables, are smooth with smooth derivatives. DNLP models can in addition use functions that are smooth but have discontinuous derivatives. The usual arithmetic operators (+, -, *, /, and **) can appear on both model classes.

The functions that can be used with endogenous arguments in a DNLP model and not in an NLP model are ABS, MIN, and MAX and as a consequence the indexed operators SMIN and SMAX.

Note that the offending functions can be applied to expressions that only involve constants such as parameters, var.l, and eq.m. Fixed variables are in principle constants, but GAMS makes its tests based on the functional form of a model, ignoring numerical parameter values and numerical bound values, and terms involving fixed variables can therefore not be used with ABS, MIN, or MAX in an NLP model.

The NLP solvers used by GAMS can also be applied to DNLP models. However, it is important to know that the NLP solvers attempt to solve the DNLP model as if it was an NLP model. The solver uses the derivatives of the constraints with respect to the variables to guide the search, and it ignores the fact that some of the derivatives may change discontinuously. There are at the moment no GAMS solvers designed specifically for DNLP models and no solvers that take into account the discontinuous nature of the derivatives in a DNLP model.

### 7.1 DNLP Models: What Can Go Wrong?

Solvers for NLP Models are all based on making marginal improvements to some initial solution until some optimality conditions ensure no direction with marginal improvements exist. A point with no marginally improving direction is called a Local Optimum.

The theory about marginal improvements is based on the assumption that the derivatives of the constraints with respect to the variables are a good approximations to the marginal changes in some neighborhood around the current point.

Consider the simple NLP model, min $SQR(x)$, where $x$ is a free variable. The marginal change in the objective is the derivative of $SQR(x)$ with respect to $x$, which is $2*x$. At $x = 0$, the marginal change in all directions is zero and $x = 0$ is therefore a Local Optimum.

Next consider the simple DNLP model, min $ABS(x)$, where $x$ again is a free variable. The marginal change in the objective is still the derivative, which is $+1$ if $x > 0$ and $-1$ if $x < 0$. When $x = 0$, the derivative depends on whether we are going to increase or decrease $x$. Internally in the DNLP solver, we cannot be sure whether the derivative at 0 will be -1 or +1; it can depend on rounding tolerances. An NLP solver will start in some initial point, say $x = 1$, and look at the derivative, here $+1$. 

Since the derivative is positive, x is reduced to reduce the objective. After some iterations, x will be zero or very close to zero. The derivative will be +1 or -1, so the solver will try to change x. However, even small changes will not lead to a better objective function. The point x = 0 does not look like a Local Optimum, even though it is a Local Optimum. The result is that the NLP solver will muddle around for some time and then stop with a message saying something like: "The solution cannot be improved, but it does not appear to be optimal."

In this first case we got the optimal solution so we can just ignore the message. However, consider the following simple two-dimensional DNLP model: $\min |x_1 + x_2| + 5|x_1 - x_2|$ with x1 and x2 free variables. Start the optimization from x1 = x2 = 1. Small increases in x1 will increase both terms and small decreases in x1 (by dx) will decrease the first term by dx but it will increase the second term by 5*dx. Any change in x1 only is therefore bad, and it is easy to see that any change in x2 only also is bad. An NLP solver may therefore be stuck in the point x1 = x2 = 1, even though it is not a local solution: the direction (dx1, dx2) = (-1, -1) will lead to the optimum in x1 = x2 = 0. However, the NLP solver cannot distinguish what happens with this model from what happened in the previous model; the message will be of the same type: "The solution cannot be improved, but it does not appear to be optimal."

### 7.2 Reformulation from DNLP to NLP

The only reliable way to solve a DNLP model is to reformulate it as an equivalent smooth NLP model. Unfortunately, it may not always be possible. In this section we will give some examples of reformulations.

The standard reformulation approach for the ABS function is to introduce positive and negative deviations as extra variables: The term $z = |f(x)|$ is replaced by $z = f_{\text{plus}} + f_{\text{minus}}$, fplus and fminus are declared as positive variables and they are defined with the identity: $f(x) = f_{\text{plus}} - f_{\text{minus}}$. The discontinuous derivative from the ABS function has disappeared and the part of the model shown here is smooth. The discontinuity has been converted into lower bounds on the new variables, but bounds are handled routinely by any NLP solver. The feasible space is larger than before; $f(x) = 5$ can be obtained both with $f_{\text{plus}} = 5$, $f_{\text{minus}} = 0$, and $z = 5$, and with $f_{\text{plus}} = 1000$, $f_{\text{minus}} = 995$, and $z = 1995$. Provided the objective function has some term that tries to minimize $z$, either fplus or fminus will become zero and $z$ will end with its proper value.

You may think that adding the smooth constraint $f_{\text{plus}} * f_{\text{minus}} = 0$ would ensure that either fplus or fminus is zero. However, this type of so-called complementarity constraint is "bad" in any NLP model. The feasible space consists of the two half lines: (fplus = 0 and fminus $\geq$ 0) and (fplus $\geq$ 0 and fminus = 0). Unfortunately, the marginal change methods used by most NLP solvers cannot move from one half line to the other, and the solution is stuck at the half line it happens to reach first.

There is also a standard reformulation approach for the MAX function. The equation $z = \max(f(x), g(y))$ is replace by the two inequalities, $z \geq f(x)$ and $z \geq g(y)$. Provided the objective function has some term that tries to minimize $z$, one of the constraints will become binding as equality and $z$ will indeed be the maximum of the two terms.

The reformulation for the MIN function is similar. The equation $z = \min(f(x), g(y))$ is replace by the two inequalities, $z \leq f(x)$ and $z \leq g(y)$. Provided the objective function has some term that tries to maximize $z$, one of the constraints will become binding as equality and $z$ is indeed the minimum of the two terms.

MAX and MIN can have more than two arguments and the extension should be obvious.

The non-smooth indexed operators, SMAX and SMIN can be handled using a similar technique: for example, $z = \max(I, f(x,I))$ is replaced by the indexed inequality: $\text{Ineq}(I) .. z = \max(I, f(x,I))$;

The reformulations that are suggested here all enlarge the feasible space. They require the objective function to move the final solution to the intersection of this larger space with the original feasible space. Unfortunately, the objective function is not always so helpful. If it is not, you may try using one of the smooth approximations described next. However, you should realize, that if the objective function cannot help the "good" approximations described here, then your overall model is definitely non-convex and it is likely to have multiple local optima.

### 7.3 Smooth Approximations

Smooth approximations to the non-smooth functions ABS, MAX, and MIN are approximations that have function values close to the original functions, but have smooth derivatives.

A smooth GAMS approximation for $|f(x)|$ is
where delta is a small scalar. The value of delta can be used to control the accuracy of the approximation and the curvature around \( f(x) = 0 \). The approximation error is largest when \( f(x) \) is zero, in which case the error is delta. The error is reduced to approximately \( \frac{\text{SQR} \cdot \text{delta}}{2} \) for \( f(x) = 1 \). The second derivative is \( 1/\text{delta} \) at \( f(x) = 0 \) (excluding terms related to the second derivative of \( f(x) \)). A delta value between \( 1.e-3 \) and \( 1.e-4 \) should in most cases be appropriate. It is possible to use a larger value in an initial optimization, reduce it and solve the model again. You should note, that if you reduce delta below \( 1.e-4 \) then large second order terms might lead to slow convergence or even prevent convergence.

The approximation shown above has its largest error when \( f(x) = 0 \) and smaller errors when \( f(x) \) is far from zero. If it is important to get accurate values of \( \text{ABS} \) exactly when \( f(x) = 0 \), then you may use the alternative approximation

\[
\text{SQRT} \left( \text{SQR} \left( f(x) \right) + \text{SQR} \left( \text{delta} \right) \right) - \text{delta}
\]

instead. The only difference is the constant term. The error is zero when \( f(x) \) is zero and the error grows to -delta when \( f(x) \) is far from zero.

Some theoretical work uses the Huber, \( H(*) \), function as an approximation for \( \text{ABS} \). The Huber function is defined as

\[
H(x) = \begin{cases} 
  x & \text{for } x > \text{delta}, \\
  -x & \text{for } x < -\text{delta} \text{ and} \\
  \text{SQR}(x)/2/\text{delta} + \text{delta}/2 & \text{for } -\text{delta} < x < \text{delta}.
\end{cases}
\]

Although the Huber function has some nice properties, it is for example accurate when \( \text{ABS}(x) > \text{delta} \), it is not so useful for GAMS work because it is defined with different formulae for the three pieces.

A smooth GAMS approximation for \( \text{MAX}(f(x),g(y)) \) is

\[
\frac{\left( f(x) + g(y) + \text{SQRT} \left( \text{SQR} \left( f(x)-g(y) \right) + \text{SQR} \left( \text{delta} \right) \right) \right)}{2}
\]

where delta again is a small scalar. The approximation error is delta/2 when \( f(x) = g(y) \) and decreases with the difference between the two terms. As before, you may subtract a constant term to shift the approximation error from the area \( f(x) = g(y) \) to areas where the difference is large. The resulting approximation becomes

\[
\frac{\left( f(x) + g(y) + \text{SQRT} \left( \text{SQR} \left( f(x)-g(y) \right) + \text{SQR} \left( \text{delta} \right) \right) \right) - \text{delta}}{2}
\]

Similar smooth GAMS approximations for \( \text{MIN}(f(x),g(y)) \) are

\[
\frac{\left( f(x) + g(y) - \text{SQRT} \left( \text{SQR} \left( f(x)-g(y) \right) + \text{SQR} \left( \text{delta} \right) \right) \right)}{2}
\]

and

\[
\frac{\left( f(x) + g(y) - \text{SQRT} \left( \text{SQR} \left( f(x)-g(y) \right) + \text{SQR} \left( \text{delta} \right) \right) + \text{delta} \right)}{2}
\]

Appropriate delta values are the same as for the \( \text{ABS} \) approximation: in the range from \( 1.e-2 \) to \( 1.e-4 \).

It appears that there are no simple symmetric extensions for \( \text{MAX} \) and \( \text{MIN} \) of three or more arguments or for indexed SMAX and SMIN.

### 7.4 Are DNLP Models Always Non-smooth?

A DNLP model is defined as a model that has an equation with an \( \text{ABS} \), \( \text{MAX} \), or \( \text{MIN} \) function with endogenous arguments. The non-smooth properties of DNLP models are derived from the non-smooth properties of these functions through the use of the chain rule. However, composite expressions involving \( \text{ABS} \), \( \text{MAX} \), or \( \text{MIN} \) can in some cases have smooth derivatives and the model can therefore in some cases be smooth.

One example of a smooth expression involving an \( \text{ABS} \) function is common in water systems modeling. The pressure loss over a pipe, \( dH \), is proportional to the flow, \( Q \), to some power, \( P \). \( P \) is usually around +2. The sign of the loss depend on the
direction of the flow so $dH$ is positive if $Q$ is positive and negative if $Q$ is negative. Although GAMS has a SIGN function, it cannot be used in a model because of its discontinuous nature. Instead, the pressure loss can be modeled with the equation

$$dH = \text{const} \times Q \times \text{ABS}(Q)^{P-1},$$

where the sign of the $Q$-term takes care of the sign of $dH$, and the ABS function guarantees that the real power $\times$ is applied to a non-negative number. Although the expression involves the ABS function, the derivatives are smooth as long as $P$ is greater than 1. The derivative with respect to $Q$ is $\text{const} \times (P-1) \times \text{ABS}(Q)^{P-1}$ for $Q > 0$ and $-\text{const} \times (P-1) \times \text{ABS}(Q)^{P-1}$ for $Q < 0$. The limit for $Q$ going to zero from both right and left is 0, so the derivative is smooth in the critical point $Q = 0$ and the overall model is therefore smooth.

Another example of a smooth expression is the following terribly looking Sigmoid expression:

$$\text{Sigmoid}(x) = \exp(\min(x,0)) / (1+\exp(-\text{abs}(x)))$$

The standard definition of the sigmoid function is

$$\text{Sigmoid}(x) = \exp(x) / (1+\exp(x))$$

This definition is well behaved for negative and small positive $x$, but it not well behaved for large positive $x$ since exp overflows. The alternative definition:

$$\text{Sigmoid}(x) = 1 / (1+\exp(-x))$$

is well behaved for positive and slightly negative $x$, but it overflows for very negative $x$. Ideally, we would like to select the first expression when $x$ is negative and the second when $x$ is positive, i.e.

$$\text{Sigmoid}(x) = \begin{cases} \exp(x)/(1+\exp(x)) & (x \lt 0) \\ (1/(1+\exp(-x))) & (x \gt 0) \end{cases}$$

but a $\text{-control}$ that depends on an endogenous variable is illegal. The first expression above solves this problem. When $x$ is negative, the nominator becomes $\exp(x)$ and the denominator becomes $1+\exp(x)$. And when $x$ is positive, the nominator becomes $\exp(0) = 1$ and the denominator becomes $1+\exp(-x)$. Since the two expressions are mathematically identical, the combined expression is of course smooth, and the exp function is never evaluated for a positive argument.

Unfortunately, GAMS cannot recognize this and similar special cases so you must always solve models with endogenous ABS, MAX, or MIN as DNLP models, even in the cases where the model is smooth.

### 7.5 Are NLP Models Always Smooth?

NLP models are defined as models in which all operators and functions are smooth. The derivatives of composite functions, that can be derived using the chain rule, will therefore in general be smooth. However, it is not always the case. The following simple composite function is not smooth: $y = \text{SQRT}(\text{SQR}(x))$. The composite function is equivalent to $y = \text{ABS}(x)$, one of the non-smooth DNLP functions.

What went wrong? The chain rule for computing derivatives of a composite function assumes that all intermediate expressions are well defined. However, the derivative of SQRT grows without bound when the argument approaches zero, violating the assumption.

There are not many cases that can lead to non-smooth composite functions, and they are all related to the case above: The real power, $x**y$, for $0 < y < 1$ and $x$ approaching zero. The SQRT function is a special case since it is equivalent to $x**y$ for $y = 0.5$.

If you have expressions involving a real power with an exponent between 0 and 1 or a SQRT, you should in most cases add bounds to your variables to ensure that the derivative or any intermediate terms used in their calculation become undefined. In the example above, $\text{SQRT}(\text{SQR}(x))$, a bound on $x$ is not possible since $x$ should be allowed to be both positive and negative. Instead, changing the expression to $\text{SQRT}(\text{SQR}(x) + \text{SQR}(\text{delta}))$ may lead to an appropriate smooth formulation.

Again, GAMS cannot recognize the potential danger in an expression involving a real power, and the presence of a real power operator is not considered enough to flag a model as a DNLP model. During the solution process, the NLP solver will compute constraint values and derivatives in various points within the bounds defined by the modeler. If these calculations result in undefined intermediate or final values, a function evaluation error is reported, an error counter is incremented, and the point is flagged as a bad point. The following action will then depend on the solver. The solver may try to continue, but
only if the modeler has allowed it with an "Option Domlim = xxx". The problem of detecting discontinuities is changed from a structural test at the GAMS model generation stage to a dynamic test during the solution process.

You may have a perfectly nice model in which intermediate terms become undefined. The composite function SQRT(POWER(x,3)) is mathematically well defined around x = 0, but the computation will involve the derivative of SQRT at zero, that is undefined. It is the modeler’s responsibility to write expressions in a way that avoids undefined intermediate terms in the function and derivatives computations. In this case, you may either add a small strictly positive lower bound on x or rewrite the function as x**1.5

8 APPENDIX A: Algorithmic Information

The objective of this Appendix is to give technically oriented users some understanding of what CONOPT is doing so they can get more information out of the iteration log. This information can be used to prevent or circumvent algorithmic difficulties or to make informed guesses about which options to experiment with to improve CONOPT’s performance on particular model classes.

8.1 Overview of GAMS/CONOPT

GAMS/CONOPT is a GRG-based algorithm specifically designed for large nonlinear programming problems expressed in the following form

\[
\begin{align*}
\text{min or max} & \quad f(x) & (1) \\
\text{subject to} & \quad g(x) = b & (2) \\
10 < x < up & & \text{lo} < x < \text{up} (3)
\end{align*}
\]

where \( x \) is the vector of optimization variables, \( \text{lo} \) and \( \text{up} \) are vectors of lower and upper bounds, some of which may be minus or plus infinity, \( b \) is a vector of right hand sides, and \( f \) and \( g \) are differentiable nonlinear functions that define the model. \( n \) will in the following denote the number of variables and \( m \) the number of equations. (2) will be referred to as the (general) constraints and (3) as the bounds.

The relationship between the mathematical model in (1)-(3) above and the GAMS model is simple: The inequalities defined in GAMS with \( =L= \) or \( =G= \) are converted into equalities by addition of properly bounded slacks. Slacks with lower and upper bound of zero are added to all GAMS equalities to ensure that the Jacobian matrix, i.e. the matrix of derivatives of the functions \( g \) with respect to the variables \( x \), has full row rank. All these slacks are together with the normal GAMS variables included in \( x \). \( \text{lo} \) represent the lower bounds defined in GAMS, either implicitly with the POSITIVE VARIABLE declaration, or explicitly with the VAR.LO notation, as well as any bounds on the slacks. Similarly, \( \text{up} \) represent upper bounds defined in GAMS, e.g. with the VAR.UP notation, as well as any bounds on the slacks. \( g \) represent the non-constant terms of the GAMS equations themselves; non-constant terms appearing on the right hand side are by GAMS moved to the left hand side and constant terms on the left hand side are moved to the right. The objective function \( f \) is simply the GAMS variable to be minimized or maximized.

Additional comments on assumptions and design criteria can be found in the Introduction to the main text.

8.2 The CONOPT Algorithm

The algorithm used in GAMS/CONOPT is based on the GRG algorithm first suggested by Abadie and Carpentier (1969). The actual implementation has many modifications to make it efficient for large models and for models written in the GAMS language. Details on the algorithm can be found in Drud (1985 and 1992). Here we will just give a short verbal description of the major steps in a generic GRG algorithm. The later sections in this Appendix will discuss some of the enhancements in CONOPT that make it possible to solve large models.

The key steps in any GRG algorithm are:

1. Initialize and Find a feasible solution.
2. Compute the Jacobian of the constraints, \( J \).
3. Select a set of \( n \) basic variables, \( x_b \), such that \( B \), the sub-matrix of basic column from \( J \), is nonsingular. Factorize \( B \). The remaining variables, \( x_n \), are called nonbasic.

4. Solve \( B^T u = df/dx_b \) for the multipliers \( u \).

5. Compute the reduced gradient, \( r = df/dx - J^T u \). \( r \) will by definition be zero for the basic variables.

6. If \( r \) projected on the bounds is small, then stop. The current point is close to optimal.

7. Select the set of superbasic variables, \( x_s \), as a subset of the nonbasic variables that profitably can be changed, and find a search direction, \( d_s \), for the superbasic variables based on \( r \) and possibly on some second order information.

8. Perform a line search along the direction \( d \). For each step, \( x_i \) is changed in the direction \( d_i \) and \( x_b \) is subsequently adjusted to satisfy \( g(x_b, x_s) = b \) in a pseudo-Newton process using the factored \( B \) from step 3.

9. Go to 2.

The individual steps are of course much more detailed in a practical implementation like CONOPT. Step 1 consists of several pre-processing steps as well as a special Phase 0 and a scaling procedure as described in the following sections Iteration 0: The Initial Point to Finding a Feasible Solution: Phase 0. The optimizing steps are specialized in several versions according to whether the model appears to be almost linear or not. For "almost" linear models some of the linear algebra work involving the matrices \( J \) and \( B \) can be avoided or done using cheap LP-type updating techniques, second order information is not relevant in step 7, and the line search in step 8 can be improved by observing that the optimal step as in LP almost always will be determined by the first variable that reaches a bound. Similarly, when the model appears to be fairly nonlinear other aspects can be optimized: the set of basic variables will often remain constant over several iterations, and other parts of the sparse matrix algebra will take advantage of this (section Finding a Feasible Solution: Phase 1 and 2 and Linear and Nonlinear Mode: Phase 1 to 4). If the model is "very" linear an improved search direction (step 7) can be computed using specialized inner LP-like iterations (section Linear Mode: The SLP Procedure) and a steepest edge procedure can be useful for certain models that needs very many iterations (section Linear Mode: The Steepest Edge Procedure). If the model is "very" nonlinear and has many degrees of freedom an improved search direction (step 7) can be computed using specialized inner SQP-like iterations based on exact second derivatives for the model (section Nonlinear Mode: The SQP Procedure).

The remaining two sections give some short guidelines for selecting non-default options (section How to Select Non-default Options), and discuss miscellaneous topics (section Miscellaneous Topics) such as CONOPT’s facilities for strictly triangular models (section Triangular Models) and for square systems of equations, in GAMS represented by the model class called CNS or Constrained Nonlinear Systems (section Constrained Nonlinear System or Square Systems of Equations), as well as numerical difficulties due to loss of feasibility (section Loss of Feasibility) and slow or no progress due to stalling (section Stalling).

### 8.3 Iteration 0: The Initial Point

The first few "iterations" in the iteration log (See section Iteration Output), in the main text for an example) are special initialization iterations, but they have been counted as real iterations to allow the user to interrupt at various stages during initialization. Iteration 0 corresponds to the input point exactly as it was received from GAMS. The sum of infeasibilities in the column labeled "Infeasibility" includes all residuals, also from the objective constraint where "Z =E= expression" will give rise to the term abs(Z - expression) that may be nonzero if Z has not been initialized. You may stop CONOPT after iteration 0 with "OPTION ITERLIM = 0;" in GAMS. The solution returned to GAMS will contain the input point and the values of the constraints in this point. The marginals of both variables and equations have not yet been computed and they will be returned as EPS.

This possibility can be used for debugging when you have a reference point that should be feasible, but is infeasible for unknown reasons. Initialize all variables to their reference values, also all intermediate variables, and call CONOPT with ITERLIM = 0. Then compute and display the following measures of infeasibility for each block of constraints, represented by the generic name EQ:

- \( \text{E} = \text{constraints: } \text{ROUND}(\text{ABS(EQ.L - EQ.LO}), 3) \)
- \( \text{L} = \text{constraints: } \text{ROUND}(\text{MIN}(0, EQ.L - EQ.UP), 3) \)
- \( \text{G} = \text{constraints: } \text{ROUND}(\text{MIN}(0, EQ.LO - EQ.L), 3) \)
The ROUND function rounds to 3 decimal places so GAMS will only display the infeasibilities that are larger than 5e-4. Similar information can be derived from inspection of the equation listing generated by GAMS with "OPTION LIMROW = nn; ", but although the method of going via CONOPT requires a little more work during implementation it can be convenient in many cases, for example for large models and for automated model checking.

8.4 Iteration 1: Preprocessing

Iteration 1 corresponds to a pre-processing step. Constraint-variable pairs that can be solved a priori (so-called pre-triangular equations and variables) are solved and the corresponding variables are assigned their final values. Constraints that always can be made feasible because they contain a free variable with a constant coefficient (so-called post-triangular equation-variable pairs) are excluded from the search for a feasible solution and from the Infeasibility measure in the iteration log. Implicitly, equations and variables are ordered as shown in Figure 1.

Preprocessing: Pre-triangular Variables and Constraints

The pre-triangular equations are those labeled A in Figure 1. They are solved one by one along the "diagonal" with respect to the pre-triangular variables labeled I. In practice, GAMS/CONOPT looks for equations with only one non-fixed variable. If such an equation exists, GAMS/CONOPT tries to solve it with respect to this non-fixed variable. If this is not possible the overall model is infeasible, and the exact reason for the infeasibility is easy to identify as shown in the examples below. Otherwise, the final value of the variable has been determined, the variable can for the rest of the optimization be considered fixed, and the equation can be removed from further consideration. The result is that the model has one equation and one non-fixed variable less. As variables are fixed new equations with only one non-fixed variable may emerge, and CONOPT repeats the process until no more equations with one non-fixed variable can be found.

This pre-processing step will often reduce the effective size of the model to be solved. Although the pre-triangular variables and equations are removed from the model during the optimization, CONOPT keeps them around until the final solution is found. The dual variables for the pre-triangular equations are then computed so they become available in GAMS.

CONOPT has a special option for analyzing and solving completely triangular models. This option is described in section Triangular Models.

The following small GAMS model shows an example of a model with pre-triangular variables and equations:

```gams
VARIABLE X1, X2, X3, OBJ;
```

PREPROCESSING: PRE-TRIANGULAR VARIABLES AND CONSTRAINTS

The pre-triangular equations are those labeled A in Figure 1. They are solved one by one along the "diagonal" with respect to the pre-triangular variables labeled I. In practice, GAMS/CONOPT looks for equations with only one non-fixed variable. If such an equation exists, GAMS/CONOPT tries to solve it with respect to this non-fixed variable. If this is not possible the overall model is infeasible, and the exact reason for the infeasibility is easy to identify as shown in the examples below. Otherwise, the final value of the variable has been determined, the variable can for the rest of the optimization be considered fixed, and the equation can be removed from further consideration. The result is that the model has one equation and one non-fixed variable less. As variables are fixed new equations with only one non-fixed variable may emerge, and CONOPT repeats the process until no more equations with one non-fixed variable can be found.

This pre-processing step will often reduce the effective size of the model to be solved. Although the pre-triangular variables and equations are removed from the model during the optimization, CONOPT keeps them around until the final solution is found. The dual variables for the pre-triangular equations are then computed so they become available in GAMS.

CONOPT has a special option for analyzing and solving completely triangular models. This option is described in section Triangular Models.

The following small GAMS model shows an example of a model with pre-triangular variables and equations:

```gams
VARIABLE X1, X2, X3, OBJ;
```
EQUATION E1, E2, E3;
E1 .. LOG(X1) + X2 =E= 1.6;
E2 .. 5 * X2 =E= 3;
E3 .. OBJ =E= SQR(X1) + 2 * SQR(X2) + 3 * SQR(X3);
X1.LO = 0.1;
MODEL DEMO / ALL /; SOLVE DEMO USING NLP MINIMIZING OBJ;

Equation E2 is first solved with respect to \( X2 \) (result \( 3/5 = 0.6 \)). It is easy to solve the equation since \( X2 \) appears linearly, and the result will be unique. \( X2 \) is then fixed and the equation is removed. Equation E1 is now a candidate since \( X1 \) is the only remaining non-fixed variable in the equation. Here \( X1 \) appears nonlinear and the value of \( X1 \) is found using an iterative scheme based on Newton’s method. The iterations are started from the value provided by the modeler or from the default initial value. In this case \( X1 \) is started from the default initial value, i.e. the lower bound of 0.1, and the result after some iterations is \( X1 = 2.718 = \exp(1) \).

During the recursive solution process it may not be possible to solve one of the equations. If the lower bound on \( X1 \) in the model above is changed to 3.0 you will get the following output:

```
** An equation in the pre-triangular part of the model cannot be solved because the critical variable is at a bound.

Residual= 9.861228876E-02
Tolerance (RTNWTR)= 6.349311267E-07

E1: Infeasibility in pre-triangular part of model.
X1: Infeasibility in pre-triangular part of model.

The solution order of the critical equations and variables is:

E2 is solved with respect to X2. Solution value = 6.0000000000E+00
E1 could not be solved with respect to X1. Final solution value = 3.0000000000E+00
E1 remains infeasible with residual = 9.8612288668E-02
```

The problem is as indicated that the variable to be solved for is at a bound, and the value suggested by Newton’s method is on the infeasible side of the bound. The critical variable is \( X1 \) and the critical equation is E1, i.e. \( X1 \) tries to exceed its bound when CONOPT solves equation E1 with respect to \( X1 \). To help you analyze the problem, especially for larger models, CONOPT reports the solution sequence that led to the infeasibility: In this case equation E2 was first solved with respect to variable \( X2 \), then equation E1 was attempted to be solved with respect to \( X1 \) at which stage the problem appeared. To make the analysis easier CONOPT will always report the minimal set of equations and variables that caused the infeasibility.

Another type of infeasibility is shown by the following model:

```
VARIABLE X1, X2, X3, OBJ;
EQUATION E1, E2, E3;
E1 .. SQR(X1) + X2 =E= 1.6;
E2 .. 5 * X2 =E= 3;
E3 .. OBJ =E= SQR(X1) + 2 * SQR(X2) + 3 * SQR(X3);
MODEL DEMO / ALL /; SOLVE DEMO USING NLP MINIMIZING OBJ;
```

where \( \log(X1) \) has been replaced by \( \sqrt{X1} \) and the lower bound on \( X1 \) has been removed. This model gives the message:

```
** An equation in the pre-triangular part of the model cannot be solved because of too small pivot.
Adding a bound or initial value may help.
```
Residual= 4.0000000
Tolerance (RTNWTR)= 6.34931126E-07

E1: Infeasibility in pre-triangular part of model.
X1: Infeasibility in pre-triangular part of model.

The solution order of the critical equations and variables is:

E2 is solved with respect to
X2. Solution value = 6.0000000000E-01

E1 could not be solved with respect to
X1. Final solution value = 0.0000000000E+00
E1 remains infeasible with residual =-4.0000000000E+00

After equation E2 has been solved with respect to X2, equation E1 that contains the term X \textsuperscript{1} \textsuperscript{2} should be solved with respect to X1. The initial value of X1 is the default value zero. The derivative of E1 with respect to X1 is therefore zero, and it is not possible for CONOPT to determine whether to increase or decrease X1. If X1 is given a nonzero initial value the model will solve. If X1 is given a positive initial value the equation will give X1 = 1, and if X1 is given a negative initial value the equation will give X1 = -1. The last type of infeasibility that can be detected during the solution of the pre-triangular or recursive equations is shown by the following example

VARIABLE X1, X2, X3, OBJ;
EQUATION E1, E2, E3, E4;
E1 .. LOG(X1) + X2 =E= 1.6;
E2 .. 5 * X2 =E= 3;
E3 .. OBJ =E= SQR(X1) + 2 * SQR(X2) + 3 * SQR(X3);
E4 .. X1 + X2 =E= 3.318;
X1.LO = 0.1;
MODEL DEMO / ALL /; SOLVE DEMO USING NLP MINIMIZING OBJ;

that is derived from the first model by the addition of equation E4. This model produces the following output

** An equation is inconsistent with other equations in the
pre-triangular part of the model.

Residual= 2.81828458E-04
Tolerance (RTNWTR)= 6.34931126E-07

The pre-triangular feasibility tolerance may be relaxed with
a line:

SET RTNWTR X.XX

in the CONOPT control program.

E4: Inconsistency in pre-triangular part of model.

The solution order of the critical equations and variables is:

E2 is solved with respect to
X2. Solution value = 6.0000000000E-01

E1 is solved with respect to
X1. Solution value = 2.7182818285E+00

All variables in equation E4 are now fixed
and the equation is infeasible. Residual = 2.8182845830E-04

First E2 is solved with respect to X2, then E1 is solved with respect to X1 as indicated by the last part of the output. At this point all variables that appear in equation E4, namely X1 and X2, are fixed, but the equation is not feasible. E4 is therefore inconsistent with E1 and E2 as indicated by the first part of the output. In this case the inconsistency is fairly small, 2.8E-04, so it could be a tolerance problem. CONOPT will always report the tolerance that was used, rtwtr - the triangular Newton tolerance, and if the infeasibility is small it will also tell how the tolerance can be relaxed. Section The CONOPT Options File gives further details on how to change tolerances, and a complete list of options is given in Appendix B.

You can turn the identification and solution of pre-triangular variables and equations off by adding the line "lspret = f" in the CONOPT control program. This can be useful in some special cases where the point defined by the pre-triangular equations gives a function evaluation error in the remaining equations. The following example shows this:

```
VARIABLE X1, X2, X3, X4, OBJ;
EQUATION E1, E2, E3, E4;
E1 .. LOG(1+X1) + X2 =E= 0;
E2 .. 5 * X2 =E= -3;
E3 .. OBJ =E= 1*SQR(X1) + 2*SQRT(0.01 + X2 - X4) + 3*SQR(X3);
E4 .. X4 =L= X2;
MODEL FER / ALL /; SOLVE FER4 MINIMIZING OBJ USING NLP;
```

All the nonlinear functions are defined in the initial point in which all variables have their default value of zero. The pre-processor will compute X2 = -0.6 from E2 and X1 = 0.822 from E1. When CONOPT continues and attempts to evaluate E3, the argument to the SQRT function is negative when these new triangular values are used together with the initial X4 = 0, and CONOPT cannot backtrack to some safe point since the function evaluation error appears the first time E3 is evaluated. When the pre-triangular preprocessor is turned off, X2 and X4 are changed at the same time and the argument to the SQRT function remains positive throughout the computations. Note, that although the purpose of the E4 inequality is to guarantee that the argument of the SQRT function is positive in all points, and although E4 is satisfied in the initial point, it is not satisfied after the pre-triangular constraints have been solved. Only simple bounds are strictly enforced at all times. Also note that if the option "lspret = f" is used then feasible linear constraints will in fact remain feasible.

An alternative (and preferable) way of avoiding the function evaluation error is to define an intermediate variable equal to 0.01+X2-X4 and add a lower bound of 0.01 on this variable. The inequality E4 could then be removed and the overall model would have the same number of constraints.

**Preprocessing: Post-triangular Variables and Constraints**

Consider the following fragment of a larger GAMS model:

```
VARIABLE UTIL(T) Utility in period T
TOTUTIL Total Utility;
EQUATION UTILDEF(T) Definition of Utility
TUTILDEF Definition of Total Utility;
UTILDEF(T).. UTIL(T) =E= nonlinear function of other variables;
TUTILDEF .. TOTUTIL =E= SUM( T, UTIL(T) / (1+R)**ORD(T) );
MODEL DEMO / ALL /; SOLVE DEMO MAXIMIZING TOTUTIL USING NLP;
```

The part of the model shown here is easy to read and from a modeling point of view it should be considered well written. However, it could be more difficult to solve than a model in which variable UTIL(T) was substituted out because all the UTILDEF equations are nonlinear constraints that the algorithms must ensure are satisfied.
To make well written models like this easy to solve CONOPT will move as many nonlinearities as possible from the constraints to the objective function. This automatically changes the model from the form that is preferable for the modeler to the form that is preferable for the algorithm. In this process CONOPT looks for free variables that only appear in one equation outside the objective function. If such a variable exists and it appears linearly in the equation, like UTIL(T) appears with coefficient 1 in equation UTILDEF(T), then the equation can always be solved with respect to the variable. This means that the variable logically can be substituted out of the model and the equation can be removed. The result is a model that has one variable and one equation less, and a more complex objective function. As variables and equations are substituted out, new candidates for elimination may emerge, so CONOPT repeats the process until no more candidates exist.

This so-called post-triangular preprocessing step will often move several nonlinear constraints into the objective function where they are much easier to handle, and the effective size of the model will decrease. In some cases the result can even be a model without any general constraints. The name post-triangular is derived from the way the equations and variables appear in the permuted Jacobian in Figure 1. The post-triangular equations and variables are the ones on the lower right hand corner labeled B and II, respectively.

In the example above, the UTIL variables will be substituted out of the model together with the nonlinear UTILDEF equations provided the UTIL variables are free and do not appear elsewhere in the model. The resulting model will have fewer nonlinear constraints, but more nonlinear terms in the objective function.

Although you may know that the nonlinear functions on the right hand side of UTILDEF always will produce positive UTIL values, you should in general not declare UTIL to be a POSITIVE VARIABLE. If you do, GAMS/CONOPT may not be able to eliminate UTIL(T), and the model will be harder to solve. It is of course unfortunate that a redundant bound changes the solution behavior, and to reduce this problem CONOPT will try to estimate the range of nonlinear expressions using interval arithmetic. If the computed range of the right hand side of the UTILDEF constraint is within the bounds of UTIL, then these bounds cannot be binding and UTIL is a so-called implied free variable that can be eliminated.

The following model fragment from a least squares model shows another case where the preprocessing step in GAMS/CONOPT is useful:

```
VARIABLE RESIDUAL(CASE) Residuals
   SSQ Sum of Squared Residuals;
EQUATION EQEST(CASE) Equation to be estimated
   SSQDEF Definition of objective;
EQEST(CASE).. RESIDUAL(CASE) =E= expression in other variables;
SSQDEF .. SSQ =E= SUM( CASE, SQR( RESIDUAL(CASE) ) );
MODEL LSQLARGE / ALL /; SOLVE LSQLARGE USING NLP MINIMIZING SSQ;
```

GAMS/CONOPT will substitute the RESIDUAL variables out of the model using the EQEST equations. The model solved by GAMS/CONOPT is therefore mathematically equivalent to the following GAMS model

```
VARIABLE SSQ Sum of Squared Residuals;
EQUATION SSQD Definition of objective;
   SSQD .. SSQ =E= SUM( CASE, SQR(expression in other variables));
MODEL LSQMALL / ALL /;
SOLVE LSQMALL USING NLP MINIMIZING SSQ;
```

However, if the "expression in other variables" is a little complicated, e.g. if it depends on several variables, then the first model, LSQLARGE, will be much faster to generate with GAMS because its derivatives in equation EQEST and SSQDEF are much simpler than the derivatives in the combined SSQD equation in the second model, LSQMALL. The larger model will therefore be faster to generate, and it will also be faster to solve because the computation of both first and second derivatives will be faster.

Note that the comments about what are good model formulations are dependent on the preprocessing capabilities in GAMS/CONOPT. Other algorithms may prefer models like LSQMALL over LSQLARGE. Also note that the variables and equations that are substituted out are still indirectly part of the model. GAMS/CONOPT evaluates the equations and computes values for the variables each time the value of the objective function is needed, and their values are available in the GAMS solution.

It is not necessary to have a coefficient of 1 for the variable to be substituted out in the post-triangular phase. However, a non-zero coefficient cannot be smaller than the absolute pivot tolerance used by CONOPT, Rtpiva.
The number of pre- and post-triangular equations and variables is printed in the log file between iteration 0 and 1 as shown in the iteration log in section Iteration Output of the main text. The sum of infeasibilities will usually decrease from iteration 0 to 1 because fewer constraints usually will be infeasible. However, it may increase as shown by the following example:

```
POSITIVE VARIABLE X, Y, Z;
EQUATION E1, E2;
  E1.. X =E= 1;
  E2.. 10*X - Y + Z =E= 0;
```

started from the default values $X.L = 0$, $Y.L = 0$, and $Z.L = 0$. The initial sum of infeasibilities is 1 (from E1 only). During pre-processing $X$ is selected as a pre-triangular variable in equation E1 and it is assigned its final value 1 so E1 becomes feasible. After this change the sum of infeasibilities increases to 10 (from E2 only).

You may stop CONOPT after iteration 1 with "OPTION ITERLIM = 1;" in GAMS. The solution returned to GAMS will contain the pre-processed values for the variables that can be assigned values from the pre-triangular equations, the computed values for the variables used to solve the post-triangular equations, and the input values for all other variables. The pre- and post-triangular constraints will be feasible, and the remaining constraints will have values that correspond to this point. The marginals of both variables and equations have not been computed yet and will be returned as EPS.

The crash procedure described in the following sub-section is an optional part of iteration 1.

**Preprocessing: The Optional Crash Procedure**

In the initial point given to CONOPT the variables are usually split into a group with initial value provided by the modeler (in the following called the assigned variables) and a group of variables for which no initial value has been provided (in the following called the default variables). The objective of the optional crash procedure is to find a point in which as many of the constraints as possible are feasible, primarily by assigning values to the default variables and by keeping the assigned variables at their initial values. The implicit assumption in this procedure is that if the modeler has assigned an initial value to a variable then this value is "better" then a default initial value.

The crash procedure is an extension of the triangular pre-processing procedure described above and is based on a simple heuristic: As long as there is an equation with only one non-fixed variable (a singleton row) then we should assign a value to the variable so the equation is satisfied or satisfied as closely as possible, and we should then temporarily fix the variable. When variables are fixed additional singleton rows may emerge and we repeat the process. When there are no singleton rows we fix one or more variables at their initial value until a singleton row appears, or until all variables have been fixed. The variables to be fixed at their initial value are selected using a heuristic that both tries to create many row singletons and tries to select variables with "good values". Since the values of many variables will come to depend in the fixed variables, the procedure favors assigned variables and among these it favors variables that appear in many feasible constraints.
CONOPT 83

Figure 2: The ordered Jacobian after Preprocessing and Crashing.

Figure 2 shows a reordered version of Figure 1. The variables labeled IV are the variables that are kept at their initial values, primarily selected from the assigned variables. The equations labeled C are then solved with respect to the variables labeled III, called the crash-triangular variables. The crash-triangular variables will often be variables without initial values, e.g. intermediate variables. The number of crash-triangular variables is shown on the iteration output between iteration 0 and 1, but only if the crash procedure is turned on.

The result of the crash procedure is an updated initial point in which usually a large number of equations will be feasible, namely all equations labeled A, B, and C in Figure 2. There is, as already shown with the small example in Section Preprocessing: Post-triangular Variables and Constraints above, no guarantee that the sum of infeasibilities will be reduced, but it is often the case, and the point will often provide a good starting point for the following procedures that finds an initial feasible solution.

The crash procedure is activated by adding the line "lstcrs=t" in the options file. The default value of lstcrs (lstcrs = Logical Switch for Triangular CRaSh) is f or false, i.e. the crash procedure is not normally used. Before turning the crash procedure on you must turn the definitional equations (see next sub-section) off.

Preprocessing: Definitional Equations

From version 3.16 CONOPT has introduced the concept of Definitional Equations. In section Simple Expressions it was recommended to introduce intermediate variables to simplify complex expressions. If the intermediate variables are free or if the bounds defined by the modeler cannot be binding then we call the constraints that define the intermediate variables "Definitional Equations" and the intermediate variables "Defined Variables". Some models have a large number of definitional equations, and CONOPT tries to recognize them and take advantage of the special structure. Defined variables that only appear in or feed into the objective are recognized as post-triangular variables as discussed in F section Preprocessing: Post-triangular Variables and Constraints but defined variables can also be used in the simultaneous constraints. The picture is similar to fig 2. with the C-rows representing the definitional constraints and the III-variables the defined variables. The main differences between crash-triangular variables and defined variables are that (1) defined variables are free or have non-binding bounds and the definitional equations can therefore always be made feasible, (2) defined variables are cheaper to recognize, (3) since they have a natural interpretation, using them is probably more numerical stable, and (4) there are most likely fewer defined variables than crash-triangular variables.

The number of definitional equations is printed in the log file between iteration 0 and 1 if CONOPT finds any. The definitional equations are used to give the defined variables new values, so it is no longer so important the give intermediate variables initial values. In the process the sum of infeasibilities may grow but CONOPT consider it more important to keep these constraints feasible. The defined variables are also made basic and they will most likely stay basic throughout the solution process.

There are three new options introduced to controls definitional equations: lusd, lusq, and lsusd. They are described in Appendix B. By default CONOPT will only look for unique definitional constraints, but the options allow the user to experiment with a more aggressive strategy.

8.5 Iteration 2: Scaling

Iteration 2 is the last dummy iteration during which the model is scaled, if scaling is turned on. The default is to turn scaling on. The Infeasibility column shows the scaled sum of infeasibilities. You may again stop CONOPT after iteration 2 with "OPTION ITERLIM = 2;" in GAMS, but the solution that is reported in GAMS will have been scaled back again so there will be no change from iteration 1 to iteration 2.

The following description of the automatic scaling procedure is included for completeness. Experiments have so far given mixed results with some advantage for scaling, and scaling is therefore by default turned on, corresponding to the CONOPT option "lascal = t". Users are recommended to be cautious with the automatic scaling procedure. If scaling is a problem, try to use manual scaling or scaling in GAMS (see section Scaling in the main text) based on an understanding of the model.

The scaling procedure multiplies all variables in group III and all constraints in group C (see Figure 1) by scale factors computed as follows:
1. CONOPT computes the largest term for each constraint, i. This is defined as the maximum of the constant right hand side, the slack (if any), and \( \text{abs}(\text{Jac}(i,j) \times X(j)) \) where \( \text{Jac}(i,j) \) is the derivative and \( X(j) \) is the variable.

2. The constraint scale factor is defined as the largest term in the constraint, projected on the interval \([Rtmins, Rtmaxs]\).
   The constraint is then divided by the constraint scale factor. Ignoring the projection, the result is a model in which the largest term in each constraint is exactly 1. The purpose of the projection is to prevent extreme scaling. The default value of \( Rtmins \) is 1 which implies that we do not scale the constraints up. Constraints with only small terms remain unchanged. The default value of \( Rtmaxs \) is \( 2^{30} \) or around \( 1.07e9 \) so terms much larger than \( 1.e9 \) are only partially scaled down and will still remain large.

3. The terms \( \text{abs}(\text{Jac}(i,j) \times X(j)) \) after constraint scaling measure the importance of each variable in the particular constraint. The variable scale is selected so the largest importance of the variable over all constraints is 1. This gives a very simple variable scale factor, namely the absolute value of the variable. The variable is then divided by the variable scale factor. To avoid extreme scaling we again project on the interval \([Rtmins, Rtmaxs]\). Variables less than \( Rtmins \) (default 1) are therefore not scaled up and variables over \( Rtmaxs \) (default \( 2^{30} = 1.07e9 \)) are only partially scaled down.

You should note that CONOPT by default scales large numbers down, but it does not scale small numbers up. You should therefore try to avoid having variables or terms in expressions that are small but significant. If this is not possible, allow CONOPT to scale up by giving the option \( rtmins \) a value less than 1.

All scale factors are rounded down to a power of 2 to preserve precision in the internal computations. To avoid difficulties with rapidly varying variables and derivatives CONOPT recomputes the scale factors at regular intervals (see \( lfscal \)).

The options that control scaling, \( lsscal \), \( lfscal \), \( rtmins \), and \( rtmaxs \), are all described in Appendix B.

### 8.6 Finding a Feasible Solution: Phase 0

The GRG algorithm used by CONOPT is a feasible path algorithm. This means that once it has found a feasible point it tries to remain feasible and follow a path of improving feasible points until it reaches a local optimum. CONOPT starts with the point provided by GAMS. This point will always satisfy the bounds (3): GAMS will simply move a variable that is outside its bounds to the nearer bound before it is presented to the solver. If the general constraints (2) also are feasible then CONOPT will work with feasible solutions throughout the optimization. However, the initial point may not satisfy the general constraints (2). If this is not the case, GAMS/CONOPT must first find an initial feasible point. This first step can be just as hard as finding an optimum for some models. For some models feasibility is the only problem.

GAMS/CONOPT has two methods for finding an initial feasible point. The first method is not very reliable but it is fast when it works; the second method is reliable but slower. The fast method is called Phase 0 and it is described in this section. It is used first. The reliable method, called Phase 1 and 2, will be used if Phase 0 terminates without a feasible solution.

Phase 0 is based on the observation that Newton’s method for solving a set of equations usually is very fast, but it may not always converge. Newton’s method in its pure form is defined for a model with the same number of variables as equations, and no bounds on the variables. With our type of model there are usually too many variables, i.e. too many degrees of freedom, and there are bounds. To get around the problem of too many variables, GAMS/CONOPT selects a subset with exactly m “basic” variables to be changed. The rest of the variables will remain fixed at their current values, that are not necessarily at bounds. To accommodate the bounds, GAMS/CONOPT will try to select variables that are away from their bounds as basic, subject to the requirement that the Basis matrix, consisting of the corresponding columns in the Jacobian, must have full rank and be well conditioned.

The Newton equations are solved to yield a vector of proposed changes for the basic variables. If the full proposed step can be applied we can hope for the fast convergence of Newton’s method. However, several things may go wrong:

1. The infeasibilities, measured by the 1-norm of \( g \) (i.e. the sum of the absolute infeasibilities, excluding the pre- and post-triangular equations), may not decrease as expected due to nonlinearities.

2. The maximum step length may have to be reduced if a basic variable otherwise would exceed one of its bounds.

In case 1. GAMS/CONOPT tries various heuristics to find a more appropriate set of basic variables. If this does not work, some "difficult" equations, i.e. equations with large infeasibilities and significant nonlinearities, are temporarily removed from the model, and Newton’s method is applied to the remaining set of "easy" equations.
In case 2, GAMS/CONOPT will remove the basic variable that first reaches one of its bounds from the basis and replace it by one of the nonbasic variables. Newton’s method is then applied to the new set of basic variables. The logic is very close to that of the dual simplex method. In cases where some of the basic variables are exactly at a bound GAMS/CONOPT uses an anti degeneracy procedure based on Ryan and Osborne (1988) to prevent cycling.

Phase 0 will end when all equations except possibly some “difficult” equations are feasible within some small tolerance. If there are no difficult equations, GAMS/CONOPT has found a feasible solution and it will proceed with Phase 3 and 4. Otherwise, Phase 1 and 2 is used to make the difficult equations feasible.

The iteration output will during Phase 0 have the following columns in the iteration log: Iter, Phase, Ninf, Infeasibility, Step, MX, and OK. The number in the Ninf column counts the number of “difficult” infeasible equations, and the number in the Infeasibility column shows the sum of the absolute infeasibilities in all the general constraints, both in the easy and in the difficult ones. There are three possible combinations of values in the MX and OK columns: combination (1) has F in the MX column and T in the OK column and it will always be combined with 1.0 in the Step column: this is an ideal Newton step. The infeasibilities in the easy equations should be reduced quickly, but the difficult equations may dominate the number in the Infeasibility column so you may not observe it. However, a few of these iterations is usually enough to terminate Phase 0. Combination (2) has T in the MX column indicating that a basic variable has reached its bound and is removed from the basis as in case 2. above. This will always be combined with T in the OK column. The Step column will show a step length less than the ideal Newton step of 1.0. Combination (3) has F in both the MX and OK column. It is the bad case and will always be combined with a step of 0.0: this is an iteration where nonlinearities are dominating and one of the heuristics from case 1. must be used.

The success of the Phase 0 procedure is based on being able to choose a good basis that will allow a full Newton step. It is therefore important that as many variables as possible have been assigned reasonable initial values so GAMS/CONOPT has some variables away from their bounds to select from. This topic was discussed in more detail in section Initial Values.

The start and the iterations of Phase 0 can, in addition to the crash option described in section Finding a Feasible Solution: Phase 0 be controlled with the three options lslack, lsmxbs, and lmmxsf described in Appendix B.

8.7 Finding a Feasible Solution: Phase 1 and 2

Most of the equations will be feasible when phase 0 stops. To remove the remaining infeasibilities CONOPT uses a procedure similar to the phase 1 procedure used in Linear Programming: artificial variables are added to the infeasible equations (the equations with Large Residuals), and the sum of these artificial variables is minimized subject to the feasible constraints remaining feasible. The artificial variable are already part of the model as slack variables; their bounds are simply relaxed temporarily.

This infeasibility minimization problem is similar to the overall optimization problem: minimize an objective function subject to equality constraints and bounds on the variables. The feasibility problem is therefore solved with the ordinary GRG optimization procedure. As the artificial variables gradually become zero, i.e. as the infeasible equations become feasible, they are taken out of the auxiliary objective function. The number of infeasibilities (shown in the Ninf column of the log file) and the sum of infeasibilities (in the Infeasibility column) will therefore both decrease monotonically.

The iteration output will label these iterations as phase 1 and/or phase 2. The distinction between phases 1 (linear mode) and 2 (nonlinear mode) is similar to the distinction between phases 3 and 4, which are described in the next sections.

8.8 Linear and Nonlinear Mode: Phase 1 to 4

The optimization itself follows step 2 to 9 of the GRG algorithm shown in The CONOPT Algorithm above. The factorization in step 3 is performed using an efficient sparse LU factorization similar to the one described by Suhl and Suhl (1990). The matrix operations in step 4 and 5 are also performed sparse.

Step 7, selection of the search direction, has several variants, depending on how nonlinear the model is locally. When the model appears to be fairly linear in the area in which the optimization is performed, i.e. when the function and constraint values are close to their linear approximation for the steps that are taken, then CONOPT takes advantages of the linearity: The derivatives (the Jacobian) are not computed in every iteration, the basis factorization is updated using cheap LP techniques as described by Reid (1982), the search direction is determined without use of second order information, i.e. similar to a steepest descend algorithm, and the initial steplength is estimated as the step length where the first variable reaches a bound; very often, this is the only step length that has to be evaluated. These cheap almost linear iterations are referred to a Linear
Mode and they are labeled Phase 1 when the model is infeasible and objective is the sum of infeasibilities and Phase 3 when
the model is feasible and the real objective function is optimized.

When the constraints and/or the objective appear to be more nonlinear CONOPT will still follow step 2 to 9 of the GRG
algorithm. However, the detailed content of each step is different. In step 2, the Jacobian must be recomputed in each iteration
since the nonlinearities imply that the derivatives change. On the other hand, the set of basic variables will often be the same
and CONOPT will take advantage of this during the factorization of the basis. In step 7 CONOPT uses the BFGS algorithm to
estimate second order information and determine search directions. And in step 8 it will often be necessary to perform more
than one step in the line search. These nonlinear iterations are labeled Phase 2 in the output if the solution is still infeasible,
and Phase 4 if it is feasible. The iterations in phase 2 and 4 are in general more expensive than the iteration in phase 1 and 3.

Some models will remain in phase 1 (linear mode) until a feasible solution is found and then continue in phase 3 until the
optimum is found, even if the model is truly nonlinear. However, most nonlinear models will have some iterations in phase 2
and/or 4 (nonlinear mode). Phase 2 and 4 indicates that the model has significant nonlinear terms around the current point:
the objective or the constraints deviate significantly from a linear model for the steps that are taken. To improve the rate of
convergence CONOPT tries to estimate second order information in the form of an estimated reduced Hessian using the
BFGS formula.

Each iteration is, in addition to the step length shown in column "Step", characterized by two logicals: MX and OK. MX
= T means that the step was maximal, i.e. it was determined by a variable reaching a bound. This is the expected value in
Phase 1 and 3. MX = F means that no variable reached a bound and the optimal step length will in general be determined by
nonlinearities. OK = T means that the line search was well-behaved and an optimal step length was found; OK = F means
that the line search was ill-behaved, which means that CONOPT would like to take a larger step, but the feasibility restoring
Newton process used during the line search did not converge for large step lengths. Iterations marked with OK = F (and
therefore also with MX = F) will usually be expensive, while iterations marked with MX = T and OK = T will be cheap.

8.9 Linear Mode: The SLP Procedure

When the model continues to appear linear CONOPT will often take many small steps, each determined by a new variable
reaching a bound. Although the line searches are fast in linear mode, each require one or more evaluations of the nonlinear
constraints, and the overall cost may become high relative to the progress. In order to avoid the many nonlinear constraint
evaluations CONOPT may replace the steepest descend direction in step 7 of the GRG algorithm with a sequential linear
programming (SLP) technique to find a search direction that anticipates the bounds on all variables and therefore gives a
larger expected change in objective in each line search. The search direction and the last basis from the SLP procedure are
used in an ordinary GRG-type line search in which the solution is made feasible at each step. The SLP procedure is only
used to generate good directions; the usual feasibility preserving steps in CONOPT are maintained, so CONOPT is still a
feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SLP-mode are identified by numbers in the column labeled "InItr" in the iteration log. The
number in the InItr column is the number of non-degenerate SLP iterations. This number is adjusted dynamically according
to the success of the previous iterations and the perceived linearity of the model.

The SLP procedure generates a scaled search direction and the expected step length in the following line search is therefore
1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model. This is indicated with MX = T and OK = T.
- The objective is nonlinear along the search direction and the optimal step is less than one. This is indicated with OK =
  T and MX = F.

CONOPT will by default determine if it should use the SLP procedure or not, based on progress information. You may turn it
off completely with the line "$l\text{esl}lp = f$" in the CONOPT options file (usually conopt.opt). The default value of $l\text{esl}lp$
($l\text{esl}lp = Logical Switch Enabling SLP mode) is t or true, i.e. the SLP procedure is enabled and CONOPT may use it
when considered appropriate. It is seldom necessary to define $l\text{esl}lp$, but it can be useful if CONOPT repeatedly turns SLP
on and off, i.e. if you see a mixture of lines in the iteration log with and without numbers in the InItr column.
8.10 Linear Mode: The Steepest Edge Procedure

When optimizing in linear mode (Phase 1 or 3) CONOPT will by default use a steepest descend algorithm to determine the search direction. CONOPT allows you to use a Steepest Edge Algorithm as an alternative. The idea, borrowed from Linear Programming, is to scale the nonbasic variables according to the Euclidean norm of the "updated column" in a standard LP tableau, the so-called edge length. A unit step for a nonbasic variable will give rise to changes in the basic variables proportional to the edge length. A unit step for a nonbasic variable with a large edge length will therefore give large changes in the basic variables which has two adverse effects relative to a unit step for a nonbasic variable with a small edge length: a basic variable is more likely to reach a bound after a very short step length, and the large change in basic variables is more likely to give rise to larger nonlinear terms.

The steepest edge algorithm has been very successful for linear programs, and our initial experience has also shown that it will give fewer iterations for most nonlinear models. However, the cost of maintaining the edge lengths can be more expensive in the nonlinear case and it depends on the model whether steepest edge results in faster overall solution times or not. CONOPT uses the updating methods for the edge lengths from LP, but it must re-initialize the edge lengths more frequently, e.g. when an inversion fails, which happens more frequently in nonlinear models than in linear models, especially in models with many product terms, e.g. blending models, where the rank of the Jacobian can change from point to point.

Steepest edge is turned on with the line, "$\text{lisanrm} = t$", in the CONOPT options file (usually \textit{conopt.opt}). The default value of $\text{lisanrm}$ ($\text{lisanrm} = \text{Logical Switch for A-NoRM}$) is \textit{f} or \textit{false}, i.e. the steepest edge procedure is turned off.

The steepest edge procedure is mainly useful during linear mode iterations. However, it has some influence in phase 2 and 4 also: The estimated reduced Hessian in the BFGS method is initialized to a diagonal matrix with elements on the diagonal computed from the edge lengths, instead of the usual scaled unit matrix.

8.11 Nonlinear Mode: The SQP Procedure

When progress is determined by nonlinearities CONOPT needs second order information. Some second order information can be derived from the line search and is used in the first iterations in Phase 2 or 4. Depending on progress, CONOPT may switch to a Sequential Quadratic Programming (SQP) procedure that works on a sub-model with linear constraints and a quadratic objective function. The constraints are a linearization of the nonlinear constraints, and the objective function is derived from the Hessian of the Lagrangian function. CONOPT will inside the SQP procedure use exact second order information computed by GAMS. The result of the SQP procedure is a search direction and a basis and CONOPT will afterwards use the same line search procedure and feasibility preserving steps as after the SLP procedure. CONOPT remains a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SQP-mode are identified by numbers in the column labeled "InItr" in the iteration log. The number in the InItr column is the number of non-degenerate SQP iterations. The effort spend inside the SQP procedure is adjusted dynamically according to the success of the previous iterations and the reduction in reduced gradient in the quadratic model.

The SQP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model of the constraints. This is indicated with MX = T and OK = T.
- The objective is much more nonlinear along the search direction than expected and the optimal step is not one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SQP procedure or not, based on progress information. You may turn it off completely with the line "$\text{lisesqp} = f$" in the CONOPT options file (usually \textit{conopt.opt}). The default value of $\text{lisesqp}$ ($\text{lisesqp} = \text{Logical Switch Enabling SQP mode}$) is \textit{t} or \textit{true}, i.e. the SQP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define $\text{lisesqp}$, but it can be used for experimentation.

In connection with 1st and 2nd derivatives the listing file (\textasteriskcentered*.lst) will have a few extra lines. The first looks as follows:

The model has 537 variables and 457 constraints
with 1597 Jacobian elements, 380 of which are nonlinear.
The Hessian of the Lagrangian has 152 elements on the diagonal, 228 elements below the diagonal, and 304 nonlinear variables.

The first two lines repeat information given in the GAMS model statistics and the last two lines describe second order information. CONOPT uses the matrix of second derivatives (the Hessian) of a linear combination of the objective and the constraints (the Lagrangian). The Hessian is symmetric and the statistics show that it has 152 elements on the diagonal and 228 below for a total of 380 elements in this case. This compares favorably to the number of elements in the matrix of first derivatives (the Jacobian).

For some models you may see the following message instead (before the usual CONOPT banner):

** Warning ** Memory Limit for Hessians exceeded.
You can use the Conopt option "rvhess"

The creation of the matrix of second derivatives has been interrupted because the matrix became too dense. A dense matrix of second derivatives will be slow to compute and it will need a lot of memory. In addition, it is likely that a dense Hessian will make some of the computations inside the SQP iterations so slow that the potential saving in number of iterations is used up computing and manipulating the Hessian.

GAMS/CONOPT can use second derivatives even if the Hessian is not available. A special version of the function evaluation routine can compute the Hessian multiplied by a vector (the so-called directional second derivative) without computing the Hessian itself. This routine is used when the Hessian is not available. The directional second derivative approach will require one directional second derivative evaluation call per inner SQP iteration instead of one Hessian evaluation per SQP sub-model.

If you get the "Memory Limit for Hessians exceeded" message you may consider rewriting some equation. Look for nonlinear functions applied to long linear or separable expressions such as log(sum(i,x(i))); as discussed in Section Simple Expressions. An expression like this will create a dense Hessian with card(i) rows and columns. You should consider introducing an intermediate variable that is equal to the long linear or separable expression and then apply the nonlinear function to this single variable. You may also experiment with allocating more memory for the dense Hessian and use it despite the higher cost; it may reduce the number of iterations. This can be done by adding the option RVhess = XX to the CONOPT options file. RVhess is a memory factor with default value 10 so you need a larger value. The value 0.0 is special; it means do not impose a memory limit on the Hessian.

The time spent on the many types of function and derivative evaluations are reported in the listing file in a section like this:

<table>
<thead>
<tr>
<th>CONOPT time Total</th>
<th>0.734 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>of which:</td>
<td></td>
</tr>
<tr>
<td>Function evaluations</td>
<td>0.031 = 4.3%</td>
</tr>
<tr>
<td>1st Derivative evaluations</td>
<td>0.020 = 2.7%</td>
</tr>
<tr>
<td>2nd Derivative evaluations</td>
<td>0.113 = 15.4%</td>
</tr>
<tr>
<td>Directional 2nd Derivative</td>
<td>0.016 = 2.1%</td>
</tr>
</tbody>
</table>

The function evaluations are computations of the nonlinear terms in the model, and 1st Derivatives evaluations are computations of the Jacobian of the model. 2nd Derivative evaluations are computations of the Hessian of the Lagrangian, and Directional 2nd derivative evaluations are computations of the Hessian multiplied by a vector, computed without computing the Hessian itself. The lines for 2nd derivatives will only be present if CONOPT has used this type of 2nd derivative.

If your model is not likely to benefit from 2nd derivative information or if you know you will run out of memory anyway you can save a small setup cost by telling GAMS/CONOPT not to generate it using option Dohess = f.

8.12 How to Select Non-default Options

The non-default options have an influence on different phases of the optimization and you must therefore first observe whether most of the time is spend in Phase 0, Phase 1 and 3, or in Phase 2 and 4.

Phase 0: The quality of Phase 0 depends on the number of iterations and on the number and sum of infeasibilities after Phase 0. The iterations in Phase 0 are much faster than the other iterations, but the overall time spend in Phase 0 may still be rather large. If this is the case, or if the infeasibilities after Phase 0 are large you may try to use the triangular crash options:
lstcrs = t

Observe if the initial sum of infeasibility after iteration 1 has been reduced, and if the number of phase 0 iterations and the number of infeasibilities at the start of phase 1 have been reduced. If lstcrs reduces the initial sum of infeasibilities but the number of iterations still is large you may try:

lslack = t

CONOPT will after the preprocessor immediately add artificial variables to all infeasible constraints so Phase 0 will be eliminated, but the sum and number of infeasibilities at the start of Phase 1 will be larger. You are in reality trading Phase 0 iterations for Phase 1 iterations.

You may also try the experimental bending line search with

lmmxsf = 1

The line search in Phase 0 will with this option be different and the infeasibilities may be reduced faster than with the default "lmmxsf = 0". It is likely to be better if the number of iterations with both MX = F and OK=\~=\~ F is large. This option may be combined with "lstcrs = t". Usually, linear constraints that are feasible will remain feasible. However, you should note that with the bending linesearch linear feasible constraints could become infeasible.

**Phase 1 and 3:** The number of iterations in Phase 1 and Phase 3 will probably be reduced if you use steepest edge, "lsanrm = t", but the overall time may increase. Steepest edge seems to be best for models with less than 5000 constraints, but work in progress tries to push this limit upwards. Try it when the number of iterations is very large, or when many iterations are poorly behaved identified with OK = F in the iteration log. The default SLP mode is usually an advantage, but it is too expensive for a few models. If you observe frequent changes between SLP mode and non-SLP mode, or if many line searches in the SLP iterations are ill-behaved with OK = F, then it may be better to turn SLP off with "lsslp = f".

**Phase 2 and 4:** There are currently not many options available if most of the time is spend in Phase 2 and Phase 4. If the change in objective during the last iterations is very small, you may reduce computer time in return for a slightly worse objective by reducing the optimality tolerance, rtredg.

### 8.13 Miscellaneous Topics

#### Triangular Models

A triangular model is one in which the non-fixed variables and the equations can be sorted such that the first equation only depends on the first variable, the second equation only depends on the first two variables, and the p-th equation only depends on the first p variables. Provided there are no difficulties with bounds or small pivots, triangular models can be solved one equation at a time using the method describe in section **Preprocessing: Pre-triangular Variables and Constraints** and the solution process will be very fast and reliable.

Triangular models can in many cases be useful for finding a good initial feasible solution: Fix a subset of the variables so the remaining model is known to be triangular and solve this triangular simulation model. Then reset the bounds on the fixed variables to their original values and solve the original model. The first solve will be very fast and if the fixed variables have been fixed at good values then the solution will also be good. The second solve will start from the good feasible solution generated by the first solve and it will usually optimize much more quickly than from a poor start.

The modeler can instruct CONOPT that a model is supposed to be triangular with the option "lstria = t". CONOPT will then use a special version of the preprocessing routine (see section **Preprocessing: Pre-triangular Variables and Constraints**) that solves the model very efficiently. If the model is solved successfully then CONOPT terminates with the message:

** Feasible solution to a recursive model.

and the Model Status will be 2, Locally Optimal, or 1, Optimal, depending on whether there were any nonlinear pivots or not. All marginals on both variables and equations are returned as 0 (zero) or EPS.

Two SOLVEs with different option files can be arranged by writing the option files as they are needed from within the GAMS program with PUT statements followed by a PUTCLOSE. You can also have two different option files, e.g., *conopt.opt* and *conopt.op2*, and select the second with the GAMS statement "<model>.optfile = 2;".
The triangular facility handles a number of error situations:

1. Non-triangular models: CONOPT will ensure that the model is indeed triangular. If it is not, CONOPT will return model status 5, Locally Infeasible, plus some information that allows the modeler to identify the mistake. The necessary information is related to the order of the variables and equations and number of occurrences of variables and equations, and since GAMS does no have a natural place for this type of information CONOPT returns it in the marginals of the equations and variables. The solution order for the triangular equations and variables that have been solved successfully are defined with positive numbers in the marginals of the equations and variables. For the remaining non-triangular variables and equations CONOPT shows the number of places they appear as negative numbers, i.e., a negative marginal for an equation shows how many of the non-triangular variables that appear in this equation. You must fix one or more variables until at least one of the non-triangular equation only has one non-fixed variable left.

2. Infeasibilities due to bounds: If some of the triangular equations cannot be solved with respect to their variable because the variable will exceed the bounds, then CONOPT will flag the equation as infeasible, keep the variable at the bound, and continue the triangular solve. The solution to the triangular model will therefore satisfy all bounds and almost all equations. The termination message will be

   ** Infeasible solution. xx artificial(s) have been introduced into the recursive equations.

   and the model status will be 5, Locally Infeasible.

   The modeler may in this case add explicit artificial variables with high costs to the infeasible constraints and the resulting point will be an initial feasible point to the overall optimization model. You will often from the mathematics of the model know that only some of the constraints can be infeasible, so you will only need to check whether to add artificials in these equations. Assume that a block of equations MATBAL(M,T) could become infeasible. Then the artificials that may be needed in this equation can be modeled and identified automatically with the following GAMS constructs:

   SET APOSART(M,T) Add a positive artificial in Matbal
   ANEGART(M,T) Add a negative artificial in Matbal;
   APOSART(M,T) = NO; ANEGART(M,T) = NO;

   POSITIVE VARIABLE
   VPOSART(M,T) Positive artificial variable in Matbal
   VNEGART(M,T) Negative artificial variable in Matbal;

   MATBAL(M,T).. Left hand side =E= right hand side
   + VPOSART(M,T)$APOSART(M,T) - VNEGART(M,T)$ANEGART(M,T);

   OBJDEF.. OBJ =E= other_terms +
   WEIGHT * SUM((M,T), VPOSART(M,T)$APOSART(M,T)
   +VNEGART(M,T)$ANEGART(M,T) );

   Solve triangular model ...

   APOSART(M,T)$(MATBAL.L(M,T) GT MATBAL.UP(M,T)) = YES;
   ANEGART(M,T)$(MATBAL.L(M,T) LT MATBAL.LO(M,T)) = YES;

   Solve final model ...

3. Small pivots: The triangular facility requires the solution of each equation to be locally unique which also means that the pivots used to solve each equation must be nonzero. The model segment

   E1 .. X1 =E= 0;
   E2 .. X1 * X2 =E= 0;

   will give the message
X2 appearing in
E2: Pivot too small for triangular model. Value=0.000E+00

** Infeasible solution. The equations were assumed to be recursive but they are not. A pivot element is too small.

However, the uniqueness of X2 may not be relevant if the solution just is going to be used as an initial point for a second model. The option "lsismp = t" (for Logical Switch: Ignore Small Pivots) will allow zero pivots as long as the corresponding equation is feasible for the given initial values.

Constrained Nonlinear System or Square Systems of Equations

There is a special model class in GAMS called CNS - Constrained Nonlinear System. A constrained nonlinear system is a square system of equations, i.e. a model in which the number of non-fixed variables is equal to the number of constraints. Currently, CONOPT and PATH are the only solvers for this model class. A CNS model can be solved with a solve statement like

    SOLVE <MODEL> USING CNS;

without an objective term. In some cases it may be convenient to solve a CNS model with a standard solve statement combined with an options file that has the statement "lssqrs = t". In the latter case, CONOPT will check that the number of non-fixed variables is equal to the number of constraints. In either case, CONOPT will attempt to solve the constraints with respect to the non-fixed variables using Newton’s method. The solution process does not include a lot of the safeguards used for ordinary NLP models and when it work it is often very fast and it uses less memory than for the corresponding NLP model. The lack of safeguards means that the solution process just will stop with an error message in some difficult situations and return the current intermediate infeasible solution. Examples of difficulties are that the Jacobian to be inverted is singular, or if one of the non-fixed variables tries to move outside their bounds as described with examples below.

Slacks in inequalities are counted as non-fixed variables which effectively means that inequalities should not be binding. Bounds on the variables are allowed, especially to prevent function evaluation errors for functions that only are defined for some arguments, but the bounds should not be binding in the final solution.

The solution returned to GAMS will in all cases have marginal values equal to 0 or EPS, both for the variables and the constraints.

The termination messages for CNS models are different from the termination messages for optimization models. The message you hope for is

    ** Feasible solution to a square system.

that usually will be combined with model status 16-Solved. If CONOPT in special cases can guarantee that the solution is unique, for example if the model is linear, then the model status will be 15-Solved Unique.

There are two potential error termination messages related to CNS models. A model with the following two constraints

    e1 .. x1 + x2 =e= 1;
    e2 .. 2*x1 + 2*x2 =e= 2;

will result in the message

    ** Error in Square System: Pivot too small.
    e2: Pivot too small.
    x1: Pivot too small.

"Pivot too small" means that the set of constraints is linearly dependent in the current point and there is no unique search direction for Newtons method so CONOPT terminates. The message points to one variable and one constraint. However, this just indicates that the linearly dependent set of constraints and variables include the constraint and variable mentioned. The offending constraint and variable will also be labeled 'DEPND' for linearly dependent in the equation listing. The error
will usually be combined with model status 5 - Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If the constraints are linearly dependent but the current point satisfy the constraints then the solution status will be 17 - Solved Singular, indicating that the point is feasible, but there is probably a whole ray of feasible solution through the current point.

It should be mentioned that the linear dependency and small pivot could be caused by the initial point and that the model could have a solution. An example is

\[
\begin{align*}
e_1 & : x_1 \times x_2 = & E = 1; \\
e_2 & : x_1 + x_2 = & E = 3; \\
x_1.1 &= 1; \ x_2.1 = 1; \\
\end{align*}
\]

A model with these two constraints and the bound

\[
\begin{align*}
e_1 & : x_1 + x_2 = & e = 2; \\
e_2 & : x_1 - x_2 = & e = 0; \\
x_1.lo &= 1.5; \\
\end{align*}
\]

will result in the message

** Error in Square System: A variable tries to exceed its bound. 
  x1: The variable tries to exceed its bound.

because the solution, \((x_1, x_2) = (1,1)\) violates the bound on \(x_1\). This error case will also be combined with model status 5-Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If you encounter problems with active bounds but you think it is caused by nonlinearities and that there is a solution, then you may try to use the bending linesearch with option "lmmxsf = t".

The CNS facility can be used to generate an initial feasible solution in almost the same way as the triangular model facility: Fix a subset of the variables so the remaining model is uniquely solvable, solve this model with the CNS solver or with \texttt{lssqrs = t}, reset the bounds on the fixed variables, and solve the original model. The CNS facility can be used on a larger class of models that include simultaneous sets of equations. However, the square system must be non-singular and feasible; CONOPT cannot, like in the triangular case, add artificial variables to some of the constraints and solve the remaining system when a variable reaches one of its bounds.

Additional information on CNS can be found at the GAMS web site: \url{http://www.gams.com/docs/document.htm}.

**Loss of Feasibility**

During the optimization you may sometimes see a phase 0 iteration and in rare cases you will see the message "Loss of Feasibility - Return to Phase 0". The background for this is as follows:

To work efficiently, CONOPT uses dynamic tolerances for feasibility and during the initial part of the optimization where the objective changes rapidly fairly large infeasibilities may be acceptable. As the change in objective in each iteration becomes smaller it will be necessary to solve the constraints more accurately so the "noise" in objective value from the inaccurate constraints will remain smaller than the real change. The noise is measured as the scalar product of the constraint residuals with the constraint marginals.

Sometimes it is necessary to revise the accuracy of the solution, for example because the algorithmic progress has slowed down or because the marginal of an inaccurate constraint has grown significantly after a basis change, e.g. when an inequality becomes binding. In these cases CONOPT will tighten the feasibility tolerance and perform one or more Newton iterations on the basic variables. This will usually be very quick and it happens silently. However, Newton’s method may fail, for example in cases where the model is degenerate and Newton tries to move a basic variable outside a bound. In this case CONOPT uses some special iteration similar to those discussed in section Finding a Feasible Solution: Phase 0 and they are labeled Phase 0.

These Phase 0 iterations may not converge, for example if the degeneracy is significant, if the model is very nonlinear locally, if the model has many product terms involving variables at zero, or if the model is poorly scaled and some constraints contain very large terms. If the iterations do not converge, CONOPT will issue the "Loss of feasibility ..." message, return to the real Phase 0 procedure, find a feasible solution with the smaller tolerance, and resume the optimization.
In rare cases you will see that CONOPT cannot find a feasible solution after the tolerances have been reduced, even though it has declared the model feasible at an earlier stage. We are working on reducing this problem. Until a final solution has been implemented you are encouraged to (1) consider if bounds on some degenerate variables can be removed, (2) look at scaling of constraints with large terms, and (3) experiment with the two feasibility tolerances, $rtnwma$ and $rtnwmi$ (see Appendix B), if this happens with your model.

**Stalling**

CONOPT will usually make steady progress towards the final solution. A degeneracy breaking strategy and the monotonicity of the objective function in other iterations should ensure that CONOPT cannot cycle. Unfortunately, there are a few places in the code where the objective function may move in the wrong direction and CONOPT may in fact cycle or move very slowly. The objective value used to compare two points, in the following called the adjusted objective value, is computed as the true objective plus a noise adjustment term equal to the scalar product of the residuals with the marginals (see section Loss of Feasibility where this noise term also is used). The noise adjustment term is very useful in allowing CONOPT to work smoothly with fairly inaccurate intermediate solutions. However, there is a disadvantage: the noise adjustment term can change even though the point itself does not change, namely when the marginals change in connection with a basis change. The adjusted objective is therefore not always monotone. When CONOPT loses feasibility and returns to Phase 0 there is an even larger chance of non-monotone behavior.

To avoid infinite loops and to allow the modeler to stop in cases with very slow progress CONOPT has an anti-stalling option. An iteration is counted as a stalled iteration if it is not degenerate and (1) the adjusted objective is worse than the best adjusted objective seen so far, or (2) the step length was zero without being degenerate (see OK = F in section Linear and Nonlinear Mode: Phase 1 to 4). CONOPT will stop if the number of consecutive stalled iterations (again not counting degenerate iterations) exceeds $lfstal$ and $lfstal$ is positive. The default value of $lfstal$ is 100. The message will be:

** Feasible solution. The tolerances are minimal and there is no change in objective although the reduced gradient is greater than the tolerance.

Large models with very flat optima can sometimes be stopped prematurely due to stalling. If it is important to find a local optimum fairly accurately then you may have to increase the value of $lfstal$.

**Overflow and NaN (Not A Number)**

Very large values or overflow can appear in a number of places. CONOPT cannot use these values in the optimization since the results would be unreliable or useless. CONOPT is therefore trying to detect these values and take appropriate action. Even though most modern computers can work with numbers from 1.e-300 to 1.e+300 CONOPT considers all numbers that are larger than 1.e40 or NaN to be ‘bad’ and useless. NaN means Not A Number and includes Infinity and Real Overflow. Bad numbers can have two sources. They can come from the modeler expressions or they can be generated internally in CONOPT. The nonlinear constraints or the derivatives of these constraints may return very large values. If a constraint returns a bad number you will see a message like

** A function value is very large or NaN (Not a Number).
Add bound to ensure that the function values are defined and bounded.

and if a derivatives returns a bad number the message will be something like

** A derivative is very large or NaN (Not a Number).
Add bound to ensure that the derivatives are defined and bounded.

The modeler must in both cases adjust the model to prevent that CONOPT receives these bad numbers so CONOPT stops immediately.

Bad numbers can also in rare cases appear as a result of computations inside CONOPT. Since all primal variables and all derivatives are bounded the bad numbers can only appear as a result of operations involving the factorization of the basis matrix, including the dual variables. You will in this case see a message like
** Overflow or Nan (Not A Number) has been encountered indicating numerical difficulties. Trying to repair by increasing the absolute and relative pivot tolerances (Rtpiva and Rvpivr) and allowing small values to be scaled up by decreasing the minimum scale factor, Rtmins.

Since the source of the problem is the factorization of the basis CONOPT will adjust the tolerances that are used while computing this factorization. Essentially, we must avoid very small pivots and the pivoting tolerances are therefore increased. The source of very small pivots can also be constraints were all terms are very small so CONOPT will also change the minimum scaling factor to try to avoid this source. CONOPT will continue the optimization after changing the tolerances and if the problem appears again if will change the tolerances even more, but since there is a limit to how much they can be adjusted CONOPT may have to give up and it will happen with this message

** Overflow or Nan (Not A Number) continues to be encountered after multiple attempt to repair by changed tolerances. CONOPT will give up.

The termination message will in this case be

** Feasible solution. The solution process has been terminated because intermediate results have become NaN (Not A Number).

or

** Feasible solution. The solution process has been terminated because intermediate results have become NaN (Not A Number).

If you encounter messages with NaN it is always a problem with scaling or structure. The most likely source is constraints with only small terms. Try to avoid them by scaling the variables and constraints appropriately, or try to use option rtmins. Or make CONOPT ignore these constraints by using a larger absolute pivot tolerance, rtpiva.

Another source are structured models with long chains of relationships. These models will usually have lags or leads and neighboring variables are related with a factor different from one, i.e.

dyn(i).. x(i) =E= x(i-1)*1.5 + u(i);

or

dif(i).. 2*x(i) =E= x(i-1) + x(i+1) + u(i);

CONOPT will try to solve the constraints in the proper order to avoid exploding variables, but a larger relative pivot tolerance, rtpivr, may also help. If you have a dynamic model and the variables are supposed to grow rapidly over time then you may consider having a scale factor that also grows with time.

** External Equations and Extrinsic Functions **

CONOPT can be used with external equations and extrinsic functions written in a programming language such as Fortran or C. Additional information is available at GAMS's web site at http://www.gams.com/docs/externalequ.htm and in the GAMS User’s Guide Appendix J.

Extrinsic functions can be debugged using the function suffixes gradn and hessn which use finite differences to approximate the gradient and Hessian using multiple function calls. For details check model drivtst in the GAMS Model Library. If external equations are present CONOPT will automatically turn the Function and Derivative Debugger on in the initial point to discover potential errors in the gradient calculation inside the external library. The debugger will not only check the gradients of the external library but even check the gradients calculated by GAMS. In rare cases the debugger may report problems in the regular algebra part of the model for which GAMS has calculated the gradients. These violates should be
small and can be ignored. After verifying that the external part of the model has been programmed correctly you may turn debugging off again by setting Lkdebg to 0 in an options file.

The debugger has two types of check. The first type ensures that the external equations do not depend on other variables than the ones you have specified in the GAMS representation. Structural errors found by these check are usually caused by programming mistakes and must be corrected. The second type of check verifies that the derivatives returned by the external equations and extrinsic functions are consistent with the rate of change in function values. A derivative is considered to be wrong if the value returned by the modeler deviates from the value computed using numerical differences by more than Rtmxj2 times the step used for the numerical difference (usually around 1.e-7). This check is correct if second derivatives are less than Rtmxj2. Rtmxj2 has a default value of 1.e4. If your model has larger second derivatives you may increase it in order not to get wrong error messages.

The number of error messages from the Function and Derivative Debugger is limited by Lfderr with a default value of 10.

### 9 APPENDIX B - Options

The options that ordinary GAMS users can access are listed below. Options starting on R assume real values, options starting on LS assume logical values (TRUE, T, 1, or FALSE, F, or 0), and all other CR-Cells starting on L assume integer values. The logical option Dohess is only used by the interface between GAMS and CONOPT.

#### 9.1 Algorithmic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF2DRV</td>
<td>Limit on errors in Directional Second Derivative evaluation.</td>
<td>10</td>
</tr>
<tr>
<td>LFDEGI</td>
<td>Limit on number of degenerate iterations before starting degeneracy breaking strategy.</td>
<td>10</td>
</tr>
<tr>
<td>LFEERR</td>
<td>Limit on number of function evaluation errors. Overwrites GAMS Domlim option</td>
<td>GAMS DomLim</td>
</tr>
<tr>
<td>LFHSOK</td>
<td>Limit on errors in Hessian evaluation.</td>
<td>10</td>
</tr>
<tr>
<td>LFITER</td>
<td>Maximum number of iterations. Overwrites GAMS Iterlim option.</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>LFMXNS</td>
<td>Maximum number of new superbasic variables added in one iteration.</td>
<td>auto</td>
</tr>
<tr>
<td>LFNICR</td>
<td>Limit on number of iterations with slow progress (relative less than Rtojbl).</td>
<td>20</td>
</tr>
<tr>
<td>LFNSUP</td>
<td>Maximum number of superbasic variables in the approximation to the Reduced Hessian.</td>
<td>auto</td>
</tr>
<tr>
<td>LFSCAL</td>
<td>Rescaling frequency.</td>
<td>5</td>
</tr>
<tr>
<td>LFSTAL</td>
<td>Limit on the number of stalled iterations.</td>
<td>100</td>
</tr>
<tr>
<td>LFUSDF</td>
<td>Limit on the number of candidates for defined variable in one constraint</td>
<td>2</td>
</tr>
<tr>
<td>LMDEBG</td>
<td>Method used by the function and derivative debugger.</td>
<td>0</td>
</tr>
<tr>
<td>LMMXSF</td>
<td>Method used to determine the step in Phase 0.</td>
<td>0</td>
</tr>
<tr>
<td>LMMXST</td>
<td>Method used to determine the maximum step while tightening tolerances.</td>
<td>0</td>
</tr>
<tr>
<td>LMNIA</td>
<td>Method for initializing the diagonal of the approximate Reduced Hessian</td>
<td>0</td>
</tr>
<tr>
<td>LMSCAL</td>
<td>Method used for scaling.</td>
<td>3</td>
</tr>
<tr>
<td>LS2NDI</td>
<td>Flag for approximating Hessian information for incoming superbasics.</td>
<td>0</td>
</tr>
<tr>
<td>LS2PTJ</td>
<td>Flag for use of perturbations to compute 2nd derivatives in SQP method.</td>
<td>1</td>
</tr>
<tr>
<td>LSANRM</td>
<td>Flag for turning Steepest Edge on.</td>
<td>0</td>
</tr>
<tr>
<td>LSCRSH</td>
<td>Flag for crashing an initial basis without fixed slacks.</td>
<td>1</td>
</tr>
<tr>
<td>LSESMLP</td>
<td>Flag for enabling SLP mode.</td>
<td>1</td>
</tr>
<tr>
<td>LSESQP</td>
<td>Flag for enabling of SQP mode.</td>
<td>1</td>
</tr>
<tr>
<td>LSISMP</td>
<td>Flag for Ignoring Small Pivots in Triangular models.</td>
<td>0</td>
</tr>
<tr>
<td>LSLACK</td>
<td>Flag for selecting initial basis as Crash-triangular variables plus slacks.</td>
<td>0</td>
</tr>
<tr>
<td>LSPOST</td>
<td>Pre-processor flag for identifying and using post-triangular equations.</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>LFDERR</td>
<td>Limit on number of error messages from function and derivative debugger.</td>
<td>10</td>
</tr>
<tr>
<td>LKDEBG</td>
<td>Flag for debugging of first derivatives</td>
<td>0</td>
</tr>
<tr>
<td>RTMXJ2</td>
<td>Upper bound on second order terms.</td>
<td>1.e4</td>
</tr>
</tbody>
</table>

### 9.2 Debugging options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSSCAL</td>
<td>Flag for dynamic scaling.</td>
<td>1</td>
</tr>
<tr>
<td>LSSQRS</td>
<td>Flag for Square System. Alternative to defining modeltype=CNS in GAMS</td>
<td>0</td>
</tr>
<tr>
<td>LSTCRS</td>
<td>Flag for using the triangular crash method.</td>
<td>0</td>
</tr>
<tr>
<td>LSTRIA</td>
<td>Flag for triangular or recursive system of equations.</td>
<td>0</td>
</tr>
<tr>
<td>LSTRID</td>
<td>Flag for turning diagnostics on for the post-triangular equations.</td>
<td>0</td>
</tr>
<tr>
<td>LSUQDF</td>
<td>Flag for requiring defined variables to be unique</td>
<td>1</td>
</tr>
<tr>
<td>LSUSDF</td>
<td>Flag for forcing defined variables into the basis</td>
<td>1</td>
</tr>
<tr>
<td>RTBND1</td>
<td>Bound filter tolerance for solution values close to a bound.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTBNDT</td>
<td>Bound tolerance for defining variables as fixed.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTIPVA</td>
<td>Absolute Pivot Tolerance for building initial basis.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTIPVR</td>
<td>Relative Pivot Tolerance for building initial basis</td>
<td>1.e-3</td>
</tr>
<tr>
<td>RTMAXJ</td>
<td>Upper bound on the value of a function value or Jacobian element</td>
<td>1.e10</td>
</tr>
<tr>
<td>RTMAXS</td>
<td>Upper bound on scale factors.</td>
<td>1.e9</td>
</tr>
<tr>
<td>RTMAXV</td>
<td>Upper bound on solution values and equation activity levels</td>
<td>1.e10</td>
</tr>
<tr>
<td>RTMINA</td>
<td>Zero filter for Jacobian elements and inversion results.</td>
<td>1.e-20</td>
</tr>
<tr>
<td>RTMINJ</td>
<td>Filter for small Jacobian elements to be ignored during scaling.</td>
<td>1.e-5</td>
</tr>
<tr>
<td>RTMNS</td>
<td>Lower bound for scale factors computed from values and 1st derivatives.</td>
<td>1</td>
</tr>
<tr>
<td>RTMNS2</td>
<td>Lower bound for scale factors based on large 2nd derivatives.</td>
<td>1.e-6</td>
</tr>
<tr>
<td>RTNWMA</td>
<td>Maximum feasibility tolerance (after scaling).</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RTNWMI</td>
<td>Minimum feasibility tolerance (after scaling).</td>
<td>4.e-10</td>
</tr>
<tr>
<td>RTNWTR</td>
<td>Feasibility tolerance for triangular equations.</td>
<td>2.0e-8</td>
</tr>
<tr>
<td>RTOBJL</td>
<td>Limit for relative change in objective for well-behaved iterations.</td>
<td>3.0e-12</td>
</tr>
<tr>
<td>RTOBJR</td>
<td>Relative accuracy of the objective function.</td>
<td>3.0e-13</td>
</tr>
<tr>
<td>RTONED</td>
<td>Accuracy of One-dimensional search.</td>
<td>0.2</td>
</tr>
<tr>
<td>RTPIVA</td>
<td>Absolute pivot tolerance.</td>
<td>1.e-10</td>
</tr>
<tr>
<td>RTPIVR</td>
<td>Relative pivot tolerance during basis factorizations.</td>
<td>0.05</td>
</tr>
<tr>
<td>RTPIVT</td>
<td>Absolute pivot tolerance for nonlinear elements in pre-triangular equations.</td>
<td>1.e-5</td>
</tr>
<tr>
<td>RTPIVU</td>
<td>Relative pivot tolerance during basis updates.</td>
<td>0.05</td>
</tr>
<tr>
<td>RTREDG</td>
<td>Optimality tolerance for reduced gradient.</td>
<td>1.e-7</td>
</tr>
<tr>
<td>RVTIME</td>
<td>Time Limit. Overwrites the GAMS Reslim option.</td>
<td>GAMS ResLim</td>
</tr>
</tbody>
</table>

### 9.3 Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFEMSG</td>
<td>Limit on number of error messages related to large function value and Jacobian elements.</td>
<td>10</td>
</tr>
<tr>
<td>LFILOG</td>
<td>Frequency for log-lines for non-SLP/SQP iterations.</td>
<td>auto</td>
</tr>
</tbody>
</table>
### 9.4 Interface options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cooptfile</td>
<td>Code file name to read options from.</td>
<td></td>
</tr>
<tr>
<td>DO2DIR</td>
<td>Flag for computing and using directional 2nd derivatives.</td>
<td>auto</td>
</tr>
<tr>
<td>DOHESS</td>
<td>Flag for computing and using 2nd derivatives as Hessian of Lagrangian.</td>
<td>auto</td>
</tr>
<tr>
<td>heaplimit</td>
<td>Maximum Heap size in MB allowed</td>
<td>Infinite</td>
</tr>
<tr>
<td>pretri2log</td>
<td>Send messages about the pre-triangular analyser to the log</td>
<td>0</td>
</tr>
<tr>
<td>RVHESS</td>
<td>Memory factor for Hessian generation: Skip if #Hessian elements &gt; #Jacobian elements*rvhess, 0 means unlimited.</td>
<td>10</td>
</tr>
</tbody>
</table>

**cooptfile (string):**

**DO2DIR (integer):** Flag for computing and using directional 2nd derivatives.

If turned on, make directional second derivatives (Hessian matrix times directional vector) available to CONOPT. If turned on, but it will be turned off of the model has external equations (defined with =X=) and the user has not provided directional second derivatives. If both the Hessian of the Lagrangian (see DOHESS) and directional second derivatives are available then CONOPT will use both: directional second derivatives are used when the expected number of iterations in the SQP sub-solver is low and the Hessian is used when the expected number of iterations is large.

(default = auto)

**DOHESS (integer):** Flag for computing and using 2nd derivatives as Hessian of Lagrangian.

If turned on, compute the structure of the Hessian of the Lagrangian and make it available to CONOPT. The default is usually on, but it will be turned off if the model has external equations (defined with =X=) or if the Hessian becomes too dense. See also DO2DIR and RVHESS.

(default = auto)

**heaplimit (real):** Maximum Heap size in MB allowed

Range: [0, maxdouble]

(default = Infinite)

**LF2DRV (integer):** Limit on errors in Directional Second Derivative evaluation.

If the evaluation of Directional Second Derivatives (Hessian information in a particular direction) has failed more than Lf2drv times CONOPT will not attempt to evaluate them any more and will switch to methods that do not use Directional Second Derivatives. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

(default = 10)

**LFDEGI (integer):** Limit on number of degenerate iterations before starting degeneracy breaking strategy.

The default CONOPT pivoting strategy has focus on numerical stability, but it can potentially cycle. When the number of consecutive degenerate iterations exceeds LFDEGI CONOPT will switch to a pivoting strategy that is guaranteed to break degeneracy but with slightly weaker numerical properties.

(default = 10)

**LFDERR (integer):** Limit on number of error messages from function and derivative debugger.

The function and derivative debugger (see LKDEBG) may find a very large number of errors, all derived from
the same source. To avoid very large amounts of output CONOPT will stop the debugger after LFDERR error have been found.

(default = 10)

**LFEERR (integer):** Limit on number of function evaluation errors. Overwrites GAMS Domlim option

Synonym: domlim

Function values and their derivatives are assumed to be defined in all points that satisfy the bounds of the model. If the function value or a derivative is not defined in a point CONOPT will try to recover by going back to a previous safe point (if one exists), but it will not do it more than at most Lfeerr times. If CONOPT is stopped by functions or derivatives not being defined it will return with an intermediate infeasible or intermediate non-optimal model status.

(default = GAMS DomLim)

**LFEMSG (integer):** Limit on number of error messages related to large function value and Jacobian elements.

Very large function value or derivatives (Jacobian elements) in a model will lead to numerical difficulties and most likely to inaccurate primal and/or dual solutions. CONOPT is therefore imposing an upper bound on the value of all function value and derivatives. This bound is 1.e30 for scaled models and RTMAXJ for unscaled models. If the bound is violated CONOPT will return with an intermediate infeasible or intermediate non-optimal solution and it will issue error messages for all the violating function value and Jacobian elements, up to a limit of Lfemsg error messages.

(default = 10)

**LFHSOK (integer):** Limit on errors in Hessian evaluation.

If the evaluation of Hessian information has failed more than Lfhsok times CONOPT will not attempt to evaluate it any more and will switch to methods that do not use the Hessian. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

(default = 10)

**LFILOG (integer):** Frequency for log-lines for non-SLP/SQP iterations.

Lfilog and Lfilos can be used to control the amount of iteration send to the log file. The non-SLP/SQP iterations, i.e. iterations in phase 0, 1, and 3, are usually fast and writing a log line for each iteration may be too much, especially for smaller models. The default value for the log frequency for these iterations is therefore set to 10 for small models, 5 for models with more than 500 constraints or 1000 variables and 1 for models with more than 2000 constraints or 3000 variables.

(default = auto)

**LFILOS (integer):** Frequency for log-lines for SLP or SQP iterations.

Lfilog and Lfilos can be used to control the amount of iteration send to the log file. Iterations using the SLP and/or SQP sub-solver, i.e. iterations in phase 2 and 4, may involve several inner iterations and the work per iteration is therefore larger than for the non-SLP/SQP iterations and it may be relevant to write log lines more frequently. The default value for the log frequency is therefore 5 for small models and 1 for models with more than 500 constraints or 1000 variables.

(default = auto)

**LFITER (integer):** Maximum number of iterations. Overwrites GAMS Iterlim option.

Synonym: iterlim

The iteration limit can be used to prevent models from spending too many resources. You should note that the cost of the different types of CONOPT iterations (phase 0 to 4) can be very different so the time limit (GAMS Reslim or option RVTIME) is often a better stopping criterion. However, the iteration limit is better for reproducing solution behavior across machines.

(default = GAMS IterLim)

**LFMXNS (integer):** Maximum number of new superbasic variables added in one iteration.
When there has been a sufficient reduction in the reduced gradient in one subspace new non-basics can be selected to enter the superbasis. The ones with largest reduced gradient of proper sign are selected, up to a limit. If Lfmxns is positive then the limit is \( \min(500, Lfmxns) \). If Lfmxns is zero (the default) then the limit is selected dynamically by CONOPT depending on model characteristics.

(default = auto)

**LFNICR (integer):** Limit on number of iterations with slow progress (relative less than RTOBJL).

The optimization is stopped if the relative change in objective is less than RTOBJL for Lfnicr consecutive well-behaved iterations.

(default = 20)

**LFNSUP (integer):** Maximum number of superbasic variables in the approximation to the Reduced Hessian.

CONOPT uses and stores a dense lower-triangular matrix as an approximation to the Reduced Hessian. The rows and columns correspond to the superbasic variable. This matrix can use a large amount of memory and computations involving the matrix can be time consuming so CONOPT imposes a limit on on the size. The limit is LFNSUP if LFNSUP is defined by the modeler and otherwise a value determined from the overall size of the model. If the number of superbasics exceeds the limit, CONOPT will switch to a method based on a combination of SQP and Conjugate Gradient iterations assuming some kind of second order information is available. If no second order information is available CONOPT will use a quasi-Newton method on a subset of the superbasic variables and rotate the subset as the reduced gradient becomes small.

(default = auto)

**LFSCAL (integer):** Rescaling frequency.

The row and column scales are recalculated at least every Lfscal new point (degenerate iterations do not count), or more frequently if conditions require it.

(default = 5)

**LFSTAL (integer):** Limit on the number of stalled iterations.

An iteration is considered a stalled iteration it there is no change in objective because the linesearch is limited by nonlinearities or numerical difficulties. Stalled iterations will have Step = 0 and F in the OK column of the log file. After a stalled iteration CONOPT will usually try various heuristics to get a better basis and a better search direction. However, the heuristics may not work as intended or they may even return to the original bad basis, so to prevent cycling CONOPT stops after Lfstal stalled iterations and returns an Intermediate Infeasible or Intermediate Nonoptimal solution.

(default = 100)

**LFTMSG (integer):** Limit on number of error messages related to infeasible pre-triangle.

If the pre-processor determines that the model is infeasible it tries to define a minimal set of variables and constraints that define the infeasibility. If this set is larger than LFTMSG elements the report is considered difficult to use and it is skipped.

(default = 25)

**LFUSDF (integer):** Limit on the number of candidates for defined variable in one constraint

When there are more than one candidate to be selected as a defined variable in a constraint CONOPT tries to select the most appropriate in order to select as many defined variables as possible. However, to avoid too much arbitrariness this is only attempted if there are not more than Lfusdf candidates.

(default = 2)

**LKDEBG (integer):** Flag for debugging of first derivatives

Lkdebg controls how often the derivatives are tested. Debugging of derivatives is only relevant for user-written functions in external equations defined with =X=. The amount of debugging is controlled by LMDEBG. See RTMXJ2 for a definition of when derivatives are considered wrong.

(default = 0)
-1 The derivatives are tested in the initial point only.

0 No debugging

+n The derivatives are tested in all iterations that can be divided by LMDEBG, provided the derivatives are computed in this iteration. (During phase 0, 1, and 3 derivatives are only computed when it appears to be necessary.)

LMDEBG (integer): Method used by the function and derivative debugger.

The function and derivative debugger (turned on with LMDEBG) can perform a fairly cheap test or a more extensive test, controlled by LMDEBG. See RTMXJ2 for a definition of when derivatives are considered wrong. All tests are performed in the current point found by the optimization algorithm.

(default = 0)

0 Perform tests for sparsity pattern and tests that the numerical values of the derivatives appear to be correct. This is the default.

1 As 0 plus make extensive test to determine if the functions and their derivatives are continuous around the current point. These tests are much more expensive and should only be used of the cheap test does not find an error but one is expected to exist.

LMMXSF (integer): Method used to determine the step in Phase 0.

The steplength used by the Newton process in phase 0 is computed by one of two alternative methods controlled by LMMXSF:

(default = 0)

0 The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.

1 A method based on bending (projecting the target values of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum. Linear constraints that initially are feasible may become infeasible due to bending. The method does not use anti-degeneracy. This will to be taken care off by removing difficult constraints from the Newton Process at regular intervals. The bending method can sometimes be useful for CNS models that stop when a variable exceeds its bound in an intermediate point even though the final solution is known to be inside the bounds.

LMMXST (integer): Method used to determine the maximum step while tightening tolerances.

The steplength used by the Newton process when tightening tolerances is computed by one of two alternative methods controlled by LMMXST:

(default = 0)

0 The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.

1 A method based on bending (projecting the target value of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum.

LMNDIA (integer): Method for initializing the diagonal of the approximate Reduced Hessian

Each time a nonbasic variable is made superbasic a new row and column is added to the approximate Reduced Hessian. The off-diagonal elements are set to zero and the diagonal to a value controlled by LMNDIA:

(default = 0)

0 The new diagonal element is set to the geometric mean of the existing diagonal elements. This gives the new diagonal element an intermediate value and new superbasic variables are therefore not given any special treatment. The initial steps should be of good size, but build-up of second order information in the new sub-space may be slower. The larger diagonal element may also in bad cases cause premature convergence.
1. The new diagonal elements is set to the minimum of the existing diagonal elements. This makes
the new diagonal element small and the importance of the new superbasic variable will therefore be
high. The initial steps can be rather small, but better second order information in the new sub-space
should be build up faster.

LMSCAL (integer): Method used for scaling.

CONOPT will by default use scaling of the equations and variables of the model to improve the numerical
behavior of the solution algorithm and the accuracy of the final solution, see LSSCAL and LMSCAL. The
objective of the scaling process is to reduce the values of all large primal and dual variables as well as the values
of all large first derivatives so they become closer to 1. Small values are usually not scaled up, see RTMAXS
and RTMINS. Scaling method 3 is recommended. The others are only kept for backward compatibility.
(default = 3)

0. Scaling is based on repeatedly dividing the rows and columns by the geometric means of the largest
and smallest elements in each row and column. Very small elements less than RTMINJ are considered
equal to RTMINJ.

1. Similar to 3 below, but the projection on the interval [Rtmins,Rtmaxs] is applied at a different stage.
With method 1, abs(X)*abs(Jac) with small X and very large Jac is scaled very aggressively with a
factor abs(Jac). With method 3, the scale factor is abs(X)*abs(Jac). The difference is seen in models
with terms like Sqrt(X) close to X = 0.

2. As 1 but the terms are computed based on a moving average of the squares X and Jac. The purpose
of the moving average is to keep the scale factor more stable. This is often an advantage, but for
models with very large terms (large variables and in particular large derivatives) in the initial point
the averaging process may not have enough time to bring the scale factors into the right region.

3. Rows are first scaled by dividing by the largest term in the row, then columns are scaled by dividing
by the maximum of the largest term and the value of the variable. A term is here defined as
abs(X)*abs(Jac) where X is the value of the variable and Jac is the value of the derivative (Jacobian
element). The scale factors are then projected on the interval between Rtmins and Rtmaxs.

LS2NDI (integer): Flag for approximating Hessian information for incoming superbasics.

If LS2NDI is turned on (1) CONOPT will try to estimate second order Hessian information for incoming superbasic
variables based on directional second derivatives. This is more costly than the standard method described under
LMNDIA.
(default = 0)

LS2PTJ (integer): Flag for use of perturbations to compute 2nd derivatives in SQP method.

If on (1) CONOPT may use perturbations of the Jacobian to compute directional 2nd derivatives if they are
not provided with other cheaper and more accurate methods. With GAMS it is only relevant for models with
functions defined in external function libraries or models with external equations defined with the =X= equation
type.
(default = 1)

LSANRM (integer): Flag for turning Steepest Edge on.

Flag used to turn steepest edge pricing on (1) or off (0). Steepest edge pricing makes the individual iterations
more expensive but it may decrease their number. Only experimentation can show if it is worth while.
(default = 0)

LSCRSH (integer): Flag for crashing an initial basis without fixed slacks

When turned on (1) CONOPT will try to crash a basis without fixed slacks in the basis. Fixed slacks are only
included in a last round to fill linearly dependent rows. When turned off, large infeasible slacks will be included
in the initial basis with preference for variables and slacks far from bound.
(default = 1)

LSESLLP (integer): Flag for enabling SLP mode.
If Lseslp is on (the default) then the SLP (sequential linear programming) sub-solver can be used, otherwise it is turned off.
(default = 1)

**LSESQP (integer):** Flag for enabling of SQP mode.

If Lsesqp is on (the default) then the SQP (sequential quadratic programming) sub-solver can be used, otherwise it is turned off.
(default = 1)

**LSISMP (integer):** Flag for ignoring small pivots in triangular models

Ignore SMall Pivots. When turned on CONOPT will ignore the non-uniqueness from small pivots during a triangular solve (see LSTRIA). Note, that the non-uniqueness may propagate to later equations, but we cannot check for it in nonlinear equations.
(default = 0)

**LSLACK (integer):** Flag for selecting initial basis as Crash-triangular variables plus slacks.

When turned on together with LSTCRS CONOPT will use the triangular crash procedure and select the initial basis as the crash-triangular variables plus slacks in all remaining rows.
(default = 0)

**LSPOST (integer):** Pre-processor flag for identifying and using post-triangular equations.

When turned on (the default) CONOPT will try to identify post-triangular equations. Otherwise this phase is ignored.
(default = 1)

**LSPRET (integer):** Pre-processor flag for identifying and using pre-triangular equations.

When turned on (the default) CONOPT will try to identify pre-triangular equations. Otherwise this phase is ignored.
(default = 1)

**LSSCAL (integer):** Flag for dynamic scaling.

When Lsscal is on (the default) CONOPT will use dynamic scaling of equations and variables. See also LMSCAL.
(default = 1)

**LSSQRS (integer):** Flag for Square System. Alternative to defining modeltype=CNS in GAMS

When turned on the modeler declares that this is a square system, i.e. the number of non-fixed variables must be equal to the number of constraints, no bounds must be active in the final solution, and the basis selected from the non-fixed variables must always be nonsingular.
(default = 0)

**LSTCRS (integer):** Flag for using the triangular crash method.

When turned on CONOPT will try to crash a triangular basis using ideas by Gould and Reid. The procedure relies on identifying and using good initial values provided by the modeler and only assigning values to variables that are not initialized. Should only be used when several important variables have been given reasonable initial values. The sum of infeasibilities may for some models grow during the crash procedure, so modelers are advised that the option should be used with caution. The option will be ignored if defined variables are forced into the bases Lsusdf.
(default = 0)

**LSTRIA (integer):** Flag for triangular or recursive system of equations.

If turned on the equations must form a recursive system. Equations that depend on known variables are allowed as long as they are consistent, e.g. \( x = 1 \) and \( 2 \times x = 2 \). If the equations are not recursive the model is considered infeasible, and the equations with minimum row count are flagged together with the columns they intersect. See
also LSISMP.
(default = 0)

**LSTRID (integer):** Flag for turning diagnostics on for the post-triangular equations.

If turned on certain diagnostic messages related to the post-triangular equations will be printed. The messages
are mainly related to unusual modeling constructs where linear variables for example only appear in the objective
or where certain constraints are guarantied redundant.

(default = 0)

**LSUQDF (integer):** Flag for requiring defined variables to be unique

When turned on (1) CONOPT will not allow defined variables unless they are unique. We exclude a variable if it
can be defined from more than one equation, and we exclude equations if they can be used to define more than
one variable.

(default = 1)

**LSUSDF (integer):** Flag for forcing defined variables into the basis

When turned on (1) CONOPT will identify defined variables from constraints of the type \( x(i) = f(x) \) where \( x(i) \) is
free or implied free. The largest number of defined variables possible will be made basic and will be assigned
initial values that are consistent with their defining constraint. When throued off (0) defined variables and their
defining constraints are treated like all other variables and constraints. When turned on the triangular crash
(LSTCRS) will not be used

(default = 1)

**pretri2log (integer):** Send messages about the pre-triangular analyser to the log

(default = 0)

**RTBND1 (real):** Bound filter tolerance for solution values close to a bound.

A variable is considered to be at a bound if its distance from the bound is less than \( \text{Rtbd1} \times \max(1, \text{ABS(Bound)}) \).
If you need a very small value then your model is probably poorly scaled.

Range: \([3.e-13, 1.e-5]\)

(default = 1.e-7)

**RTBNDT (real):** Bound tolerance for defining variables as fixed.

A variable is considered fixed if the distance between the bounds is less than \( \text{Rtbdnt} \times \max(1, \text{Abs(Bound)}) \).
The tolerance is also used on implied bounds (from converted inequalities) and these implied bounds may be
infeasible up to \( \text{Rtbdnt} \).

Range: \([3.e-13, 1.e-5]\)

(default = 1.e-7)

**RTIPVA (real):** Absolute Pivot Tolerance for building initial basis.

Absolute pivot tolerance used during the search for a first logically non-singular basis. The default is fairly large
to encourage a better conditioned initial basis.

Range: \([3.e-13, 1.e-3]\)

(default = 1.e-7)

**RTIPVR (real):** Relative Pivot Tolerance for building initial basis

Relative pivot tolerance used during the search for a first logically non-singular basis.

Range: \([1.e-4, 0.9]\)

(default = 1.e-3)

**RTMAXJ (real):** Upper bound on the value of a function value or Jacobian element.

Very large values of variables, function value, and derivatives and in particular large variations in the absolute
value of the variables, functions, and derivatives makes the model harder to solve and poses problems for both feasibility and optimality tests. CONOPT will usually try to scale the model (see LSSCAL) to remove these problems. However, scaling can also make important aspects of a model appear un-important and there is therefore a limit to how aggressively we can scale a model (see RTMAXS and RTMINS). To avoid serious scaling problems CONOPT poses upper bounds on all variables (see RTMAXV) and all function value and derivatives, RTMAXJ.

Range: $[1.e4, 1.e30]$
(default = $1.e10$)

RTMAXS (real): Upper bound on scale factors.

Scale factors are projected on the interval from Rtmins to Rtmaxs. Is used to prevent very large or very small scale factors due to pathological types of constraints. RTMAXS is silently increased to $\max(\text{RTMAXV},\text{RTMAXS})/100$ if RTMAXV or RTMAXJ have large non-default values.

Range: $[1, 1.e20]$
(default = $1.e9$)

RTMAXV (real): Upper bound on solution values and equation activity levels

See RTMAXJ for a discussion of why CONOPT poses upper bounds on variables and derivatives. If the value of a variable, including the objective function value, exceeds RTMAXV then the model is considered to be unbounded and the optimization process returns the solution with the large variable flagged as unbounded.

Range: $[1.e5, 1.e30]$
(default = $1.e10$)

RTMINA (real): Zero filter for Jacobian elements and inversion results.

Contains the smallest absolute value that an intermediate result can have. If it is smaller, it is set to zero. It must be smaller than RTPIVA/10.

(default = $1.e-20$)

RTMINJ (real): Filter for small Jacobian elements to be ignored during scaling.

A Jacobian element is considered insignificant if it is less than Rtminj. The value is used to select which small values are scaled up during scaling of the Jacobian.

Range: $[1.e-7, 1.e-3]$
(default = $1.e-5$)

RTMINS (real): Lower bound for scale factors computed from values and 1st derivatives.

Scale factors used to scale variables and equations are projected on the range Rtmins to Rtmaxs. The limits are used to prevent very large or very small scale factors due to pathological types of constraints. The default value for Rtmins is 1 which means that small values are not scaled up. If you need to scale small value up towards 1 then you must define a value of Rtmins < 1.

Range: $[1.e-10, 1]$
(default = 1)

RTMNS2 (real): Lower bound for scale factors based on large 2nd derivatives.

Scaling of the model is in most cases based on the values of the variables and the first derivatives. However, if the scaled variables and derivatives are reasonable but there are large values in the Hessian of the Lagrangian (the matrix of 2nd derivatives) then the lower bound on the scale factor can be made smaller than Rtmins. CONOPT will try to scale variables with large 2nd derivatives by one over the square root of the diagonal elements of the Hessian. However, the revised scale factors cannot be less than Rtmns2.

Range: $[1.e-9, 1]$
(default = $1.e-6$)

RTMXJ2 (real): Upper bound on second order terms.
The function and derivative debugger (see \texttt{LKDEBG}) tests if derivatives computed using the modelers routine are sufficiently close to the values computed using finite differences. The term for the acceptable difference includes a second order term and uses \texttt{RTMXJ2} as an upper bound on second order derivatives in the model. Larger \texttt{RTMXJ2} values will allow larger deviations between the user-defined derivatives and the numerically computed derivatives.

\texttt{(default = 1.e4)}

\textbf{RTNWMA (real):} Maximum feasibility tolerance (after scaling).

The feasibility tolerance used by CONOPT is dynamic. As long as we are far from the optimal solution and make large steps it is not necessary to compute intermediate solutions very accurately. When we approach the optimum and make smaller steps we need more accuracy. \texttt{RTNWMA} is the upper bound on the dynamic feasibility tolerance and \texttt{RTNWMI} is the lower bound.

Range: $[1.e-10, 1.e-3]$

\texttt{(default = 1.e-7)}

\textbf{RTNWMI (real):} Minimum feasibility tolerance (after scaling).

See \texttt{RTNWMA} for a discussion of the dynamic feasibility tolerances used by CONOPT.

Range: $[4.e-11, 1.e-5]$

\texttt{(default = 4.e-10)}

\textbf{RTNWTR (real):} Feasibility tolerance for triangular equations.

Triangular equations are usually solved to an accuracy of \texttt{RTNWMI}. However, if a variable reaches a bound or a constraint only has pre-determined variables then the feasibility tolerance can be relaxed to \texttt{Rtnwtr}.

Range: $[3.e-13, 1.e-4]$

\texttt{(default = 2.0e-8)}

\textbf{RTOBJL (real):} Limit for relative change in objective for well-behaved iterations.

The change in objective in a well-behaved iteration is considered small and the iteration counts as slow progress if the change is less than $\texttt{Rtobjl} \times \max(1, \text{Abs(Objective)})$. See also \texttt{LFNICR}.

Range: $[3.0e-13, 1.0e-5]$

\texttt{(default = 3.0e-12)}

\textbf{RTOBJR (real):} Relative accuracy of the objective function.

It is assumed that the objective function can be computed to an accuracy of $\texttt{Rtobjr} \times \max(1, \text{abs(Objective)})$. Smaller changes in objective are considered to be caused by round-off errors.

Range: $[3.0e-14, 1.0e-6]$

\texttt{(default = 3.0e-13)}

\textbf{RTONED (real):} Accuracy of One-dimensional search.

The onedimensional search is stopped if the expected decrease in then objective estimated from a quadratic approximation is less than $\texttt{Rtoned}$ times the decrease so far in this onedimensional search.

Range: $[0.05, 0.8]$

\texttt{(default = 0.2)}

\textbf{RTPIVA (real):} Absolute pivot tolerance.

During LU-factorization of the basis matrix a pivot element is considered large enough if its absolute value is larger than \texttt{Rtpiva}. There is also a relative test, see \texttt{RTPIVR}.

Range: $[2.2e-16, 1.e-7]$

\texttt{(default = 1.e-10)}

\textbf{RTPIVR (real):} Relative pivot tolerance during basis factorizations.
During LU-factorization of the basis matrix a pivot element is considered large enough relative to other elements in the column if its absolute value is at least $R_{tpivr} \times$ the largest absolute value in the column. Small values or $R_{tpivr}$ will often give a sparser basis factorization at the expense of the numerical accuracy. The value used internally is therefore adjusted dynamically between the users value and 0.9, based on various statistics collected during the solution process. Certain models derived from finite element approximations of partial differential equations can give rise to poor numerical accuracy and a larger user-value of $R_{tpivr}$ may help.

Range: $[1.e^{-3}, 0.9]$

(default = 0.05)

**RTPIVT (real):** Absolute pivot tolerance for nonlinear elements in pre-triangular equations.

The smallest pivot that can be used for nonlinear or variable Jacobian elements during the pre-triangular solve. The pivot tolerance for linear or constant Jacobian elements is $R_{tpiva}$. The value cannot be less that $R_{tpiva}$.

Range: $[2.2e^{-16}, 1.e^{-3}]$

(default = 1.e^{-5})

**RTPIVU (real):** Relative pivot tolerance during basis updates.

During basis changes CONOPT attempts to use cheap updates of the LU-factors of the basis. A pivot is considered large enough relative to the alternatives in the column if its absolute value is at least $R_{tpivu} \times$ the other element. Smaller values of $R_{tpivu}$ will allow sparser basis updates but may cause accumulation of larger numerical errors.

Range: $[1.e^{-3}, 0.9]$

(default = 0.05)

**RTREDG (real):** Optimality tolerance for reduced gradient.

The reduced gradient is considered zero and the solution optimal if the largest superbasic component of the reduced gradient is less than $R_{tredg}$.

Range: $[3.e^{-13}, 1]$

(default = 1.e^{-7})

**RVHESS (real):** Memory factor for Hessian generation: Skip if $\#$Hessian elements $> \#$Jacobian elements$\times$Rvhes, 0 means unlimited.

The Hessian of the Lagrangian is considered too dense and is not passed on to CONOPT if the number of nonzero elements in the Hessian of the Lagrangian is greater than the number of nonlinear Jacobian elements multiplied by Rvhes. The assumption is that a very dense Hessian is expensive both to compute and use. If Rvhes = 0.0 then there is no limit on the number of Hessian elements.

(default = 10)

**RVTIME (real):** Time Limit. Overwrites the GAMS Reslim option.

Synonym: reslim

The upper bound on the total number of seconds that can be used in the execution phase. There are only tests for time limit once per iteration. The default value is 10000. Rvtime is overwritten by Reslim when called from GAMS. Rvtime is defined in ProbSize and/or UpdtSize when used as a subroutine library.

Range: $[0, \text{maxdouble}]$

(default = GAMS ResLim)

10 APPENDIX C: References


1 Introduction

GAMS/Cplex is a GAMS solver that allows users to combine the high level modeling capabilities of GAMS with the power of Cplex optimizers. Cplex optimizers are designed to solve large, difficult problems quickly and with minimal user intervention.
Access is provided (subject to proper licensing) to Cplex solution algorithms for linear, quadratically constrained and mixed integer programming problems. While numerous solving options are available, GAMS/Cplex automatically calculates and sets most options at the best values for specific problems.

All Cplex options available through GAMS/Cplex are summarized at the end of this document.

2 How to Run a Model with Cplex

The following statement can be used inside your GAMS program to specify using Cplex

```
Option LP = Cplex;  \{ or QCP, MIP, MIQCP, RMIP or RMIQCP \}
```

The above statement should appear before the Solve statement. The MIP and QCP capabilities are separately licensed, so you may not be able to use Cplex for those problem types on your system. If Cplex was specified as the default solver during GAMS installation, the above statement is not necessary.

3 Overview of Cplex

3.1 Linear Programming

Cplex solves LP problems using several alternative algorithms. The majority of LP problems solve best using Cplex’s state of the art dual simplex algorithm. Certain types of problems benefit from using the primal simplex algorithm, the network optimizer, the barrier algorithm, or the sifting algorithm. The concurrent option will allow solving with different algorithms in parallel. The solution is returned by the first to finish.

Solving linear programming problems is memory intensive. Even though Cplex manages memory very efficiently, insufficient physical memory is one of the most common problems when running large LPs. When memory is limited, Cplex will automatically make adjustments which may negatively impact performance. If you are working with large models, study the section entitled Physical Memory Limitations carefully.

Cplex is designed to solve the majority of LP problems using default option settings. These settings usually provide the best overall problem optimization speed and reliability. However, there are occasionally reasons for changing option settings to improve performance, avoid numerical difficulties, control optimization run duration, or control output options.

Some problems solve faster with the primal simplex algorithm rather than the default dual simplex algorithm. Very few problems exhibit poor numerical performance in both the primal and the dual. Therefore, consider trying primal simplex if numerical problems occur while using dual simplex.

Cplex has a very efficient algorithm for network models. Network constraints have the following property:

- each non-zero coefficient is either a +1 or a -1
- each column appearing in these constraints has > exactly 2 nonzero entries, one with a +1 coefficient and one with a -1 coefficient

Cplex can also automatically extract networks that do not adhere to the above conventions as long as they can be transformed to have those properties.

The barrier algorithm is an alternative to the simplex method for solving linear programs. It employs a primal-dual logarithmic barrier algorithm which generates a sequence of strictly positive primal and dual solutions. Specifying the barrier algorithm may be advantageous for large, sparse problems.

Cplex provides a sifting algorithm which can be effective on problems with many more variables than equations. Sifting solves a sequence of LP subproblems where the results from one subproblem are used to select columns from the original model for inclusion in the next subproblem.

GAMS/Cplex also provides access to the Cplex Infeasibility Finder. The Infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is
infeasible but becomes feasible if any one member of the set is dropped. GAMS/Cplex reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing. IIS is available for LP problems only.

### 3.2 Quadratically Constrained Programming

Cplex can solve models with quadratic contraints. These are formulated in GAMS as models of type QCP. QCP models are solved with the Cplex Barrier method.

QP models are a special case that can be reformulated to have a quadratic objective function and only linear constraints. Those are automatically reformulated from GAMS QCP models and can be solved with any of the Cplex QP methods (Barrier, Primal Simplex or Dual Simplex).

For QCP models, Cplex returns a primal only solution to GAMS. Dual values are returned for QP models.

### 3.3 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with integer variables, Cplex uses a branch and cut algorithm which solves a series of LP, subproblems. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS and GAMS/Cplex support Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

Cplex can also solve problems of GAMS model type MIQCP. As in the continuous case, if the base model is a QP the Simplex methods can be used and duals will be available at the solution. If the base model is a QCP, only the Barrier method can be used for the nodes and only primal values will be available at the solution.

### 3.4 Feasible Relaxation

The Infeasibility Finder identifies the causes of infeasibility by means of inconsistent set of constraints (IIS). However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Cplex is known as FeasOpt (for Feasible Optimization) and turned on by parameter feasopt in a CPLEX option file. More details can be found in the section entitled Using the Feasibility Relaxation.

### 3.5 Solution Pool: Generating and Keeping

Multiple Solutions

This chapter introduces the solution pool for storing multiple solutions to a mixed integer programming problem (MIP and MIQCP). The chapter also explains techniques for generating and managing those solutions.

The solution pool stores multiple solutions to a mixed integer programming (MIP and MIQCP) model. With this feature, you can direct the algorithm to generate multiple solutions in addition to the optimal solution. For example, some constraints may be difficult to formulate efficiently as linear expressions, or the objective may be difficult to quantify exactly. In such cases, obtaining multiple solutions will help you choose one which best fits all your criteria, including the criteria that could not be expressed easily in a conventional MIP or MIQCP model. For example,

- You can collect solutions within a given percentage of the optimal solution. To do so, apply the solution pool gap parameters solnpooleqgap and solnpoolgap.
• You can collect a set of diverse solutions. To do so, use the solution pool replacement parameter `SolnPool\-Replace` to set the solution pool replacement strategy to 2. In order to control the diversity of solutions even more finely, apply a `diversityfilter`.

• In an advanced application of this feature, you can collect solutions with specific properties. To do so, see the use of the `incumbent filter`.

• You can collect all solutions or all optimal solutions to model. To do so, set the solution pool intensity parameter `SolnPoolIntensity` to its highest value.

Please note, that the value for `best possible` can exceed the optimal solution value if CPLEX has already solved the model to optimality but continues to search for additional solutions.

Filling the Solution Pool

There are two ways to fill the solution pool associated with a model: You can accumulate successive incumbents or generate alternative solutions by populating the solution pool. The method is selected with the parameter `SolnPoolPop`:

• The regular optimization procedure automatically adds incumbents to the solution pool as they are discovered (`SolnPoolPop=1`).

• Cplex also provides a procedure specifically to generate multiple solutions. You can invoke this procedure by setting option `SolnPoolPop=2`. You can also invoke this procedure many times in a row in order to explore the solution space differently. In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory. This is done by specifying a GAMS program (option `SolnPoolPop\-repeat`) that inspects the solutions. In case this GAMS program terminates normally, i.e. no execution or compilation error, the exploration for alternative solutions proceeds.

The option `SolnPoolReplace` designates the strategy for replacing a solution in the solution pool when the solution pool has reached its capacity. The value 0 replaces solutions according to a first-in, first-out policy. The value 1 keeps the solutions with the best objective values. The value 2 replaces solutions in order to build a set of diverse solutions.

If the solutions you obtain are too similar to each other, try setting `SolnPoolReplace` to 2.

The replacement strategy applies only to the subset of solutions created in the current call of populate. Solutions already in the pool are not affected by the replacement strategy. They will not be replaced, even if they satisfy the criterion of the replacement strategy. So with every repeated call of the populate procedure the solution pool will be extended by the newly found solution. After the GAMS program specified in `SolnPoolPop\-repeat` determined to continue the search for alternative solutions, the file specified by option `SolnPoolPop\-del` option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

Details can be found in the model `solnpool` in the GAMS model library.

Enumerating All Solutions

With the solution pool, you can collect all solutions to a model. To do so, set the solution pool intensity parameter `SolnPoolIntensity` to its highest value, 4 and set `SolnPoolPop=2`.

You can also enumerate all solutions that are valid for a specific criterion. For example, if you want to enumerate all alternative optimal solutions, do the following:

• Set the pool absolute gap parameter `SolnPoolAGap=0.0`.

• Set the pool intensity parameter `SolnPoolIntensity=4`.

• Set the populate limit parameter `PopulateLim` to a value sufficiently large for your model; for example, 2100000000.

• Set the pool population parameter `SolnPoolPop=2`.
Beware, however, that, even for small models, the number of possible solutions is likely to be huge. Consequently, enumerating all of them will take time and consume a large quantity of memory.

There may be an infinite number of possible values for a continuous variable, and it is not practical to enumerate all of them on a finite-precision computer. Therefore, populate gives only one solution for each set of binary and integer variables, even though there may exist several solutions that have the same values for all binary and integer variables but different values for continuous variables.

Likewise, for the same reason, the populate procedure does not generate all possible solutions for unbounded models. As soon as the proof of unboundedness is obtained, the populate procedure stops.

Cplex uses numerical methods of finite-precision arithmetic. Consequently, the feasibility of a solution depends on the value given to tolerances. Two parameters define the tolerances that assess the feasibility of a solution:

- the integrality tolerance $E_{p\text{Int}}$
- the feasibility tolerance $E_{p\text{RHS}}$

A solution may be considered feasible for one pair of values for these two parameters, and infeasible for a different pair. This phenomenon is especially noticeable in models with numeric difficulties, for example, in models with Big M coefficients.

Since the definition of a feasible solution is subject to tolerances, the total number of solutions to a model may vary, depending on the approach used to enumerate solutions, and on precisely which tolerances are used. In most models, this tolerance issue is not problematic. But, in the presence of numeric difficulties, Cplex may create solutions that are slightly infeasible or integer infeasible, and therefore create more solutions than expected.

Filtering the Solution Pool

Filtering allows you to control properties of the solutions generated and stored in the solution pool. Cplex provides two predefined ways to filter solutions.

If you want to filter solutions based on their difference as compared to a reference solution, use a diversity filter. This filter is practical for most purposes. However, if you require finer control of which solutions to keep and which to eliminate, use the incumbent filter.

Diversity Filter

A diversity filter allows you to generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables using dot option divflt and lower and upper bounds divfltlo and divfltup. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution. If you need more than one diversity filter, for example, to generate solutions that share the characteristics of several different solutions, additional filters can be specified through a Cplex Filter File using parameter ReadFLT. Details can be found in the example model solnpool in the GAMS model library.

Incumbent Filter

If you need to enforce more complex constraints on solutions (e.g. if you need to enforce nonlinear constraints), you can use the incumbent filtering. The incumbent checking routine is part of the GAMS BCH Facility. It will accept or reject incumbents independent of a solution pool. During the populate or regular optimize procedure, the incumbent checking routine specified by the parameter userincbcall is called each time a new solution is found, even if the new solution does not improve the objective value of the incumbent. The incumbent filter allows your application to accept or reject the new solution based on your own criteria. If the GAMS program specified by userincbcall terminates normally, the solution is rejected. If this program returns with a compilation or execution error, the incumbent is accepted.

Accessing the Solution Pool

The GAMS/Cplex link produces, if properly instructed, a GDX file with name specified in SolnPool that contains a set Index with elements file1, file2, ... The associated text of these elements contain the file names of the individual GDX
solution file. The name is constructed using the prefix soln (which can be specified differently by option SolnPoolPrefix), the name of the model and a sequence number. For example soln\_loc\_p1.gdx. GAMS/Cplex will overwrite existing GDX files without warning. The set Index allows us to conveniently walk through the different solutions in the solution pool:

```plaintext
... solve mymodel min z using mip;
set soln possible solutions in the solution pool /file1*file1000/
   solnpool(soln) actual solutions;
file fsol;
execute_load 'solnpool.gdx', solnpool=Index;
loop(solnpool(soln),
   put_utility fsol 'gdxin' / solnpool.te(soln):0:0;
   execute_loadpoint;
   display z.l;
);
```

### 3.6 Cplex Remote Object Server and Distributed MIP

The Cplex Remote Object Server allows you to use a server to offload all of your Cplex computations. Cplex Remote Object Server licenses and software are not included in GAMS/Cplex. You need to get in contact with IBM. GAMS/Cplex merely supports the use of a Cplex Remote Object Server. You can specify the server with the ComputeServer option.


### 4 GAMS Options

The following GAMS options are used by GAMS/Cplex:

- **Option BRatio = x;**
  
  Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Cplex not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed.

- **Option IterLim = n;**
  
  Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution. Cplex handles the iteration limit for MIP problems differently than some other GAMS solvers. The iteration limit is applied per node instead of as a total over all nodes. For MIP problems, controlling the length of the solution run by limiting the execution time (ResLim) is preferable.

  Similarly, when using the sifting algorithm, the iteration limit is applied per sifting iteration (ie per LP). The number of sifting iterations (LPs) can be limited by setting Cplex parameter siftitlim. It is the number of sifting iterations that is reported back to GAMS as iterations used.

- **Option ResLim = x;**
  
  Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. In case a pre-solve is done, the post-solve routine will be invoked before reporting the solution.
• **Option SysOut = On;**
  Will echo Cplex messages to the GAMS listing file. This option may be useful in case of a solver failure.

• **ModelName.Cheat = x;**
  Cheat value: each new integer solution must be at least x better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the OptCA option). The Cplex option objdif overrides the GAMS cheat parameter.

• **ModelName.Cutoff = x;**
  Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm.

• **ModelName.NodLim = x;**
  Maximum number of nodes to process for a MIP problem.

• **ModelName.OptCA = x;**
  Absolute optimality criterion for a MIP problem.

• **ModelName.OptCR = x;**
  Relative optimality criterion for a MIP problem. Notice that Cplex uses a different definition than GAMS normally uses. The OptCR option asks Cplex to stop when
  \[
  \left(\frac{|BP - BF|}{1.0e-10 + |BF|}\right) < \text{OptCR}
  \]
  where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. The GAMS definition is:
  \[
  \left(\frac{|BP - BF|}{|BP|}\right) < \text{OptCR}
  \]

• **ModelName.OptFile= 1;**
  Instructs Cplex to read the option file. The name of the option file is cplex.opt.

• **ModelName.PriorOpt= 1;**
  Instructs Cplex to use priority branching information passed by GAMS through the variable.prior' parameters.

• **ModelName.TryInt= x;**
  Causes GAMS/Cplex to make use of current variable values when solving a MIP problem. If a variable value is within x of a bound, it will be moved to the bound and the preferred branching direction for that variable will be set toward the bound. The preferred branching direction will only be effective when priorities are used. Priorities and tryint are sometimes not very effective and often outperformed by GAMS/CPLEX default settings. Supporting GAMS/CPLEX with knowledge about a known solution can be passed on by different means, please read more about this in section entitled Starting from a MIP Solution.

## 5 Summary of CPLEX Options

The various Cplex options are listed here by category, with a few words about each to indicate its function. The options are listed again, in alphabetical order and with detailed descriptions, in the last section of this document.
## 5.1 Preprocessing and General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>advind</td>
<td>advanced basis use</td>
<td>determined by GAMS Bratio</td>
</tr>
<tr>
<td>aggfill</td>
<td>aggregator fill parameter</td>
<td>10</td>
</tr>
<tr>
<td>aggind</td>
<td>aggregator on/off</td>
<td>-1</td>
</tr>
<tr>
<td>calcqcpduals</td>
<td>calculate the dual values of a quadratically constrained problem</td>
<td>1</td>
</tr>
<tr>
<td>clocktype</td>
<td>clock type for computation time</td>
<td>0</td>
</tr>
<tr>
<td>coeredind</td>
<td>coefficient reduction on/off</td>
<td>-1</td>
</tr>
<tr>
<td>computeserver</td>
<td>address and port of Cplex remote object server</td>
<td></td>
</tr>
<tr>
<td>depind</td>
<td>dependency checker on/off</td>
<td>-1</td>
</tr>
<tr>
<td>dettilim</td>
<td>deterministic time limit</td>
<td>1e+075</td>
</tr>
<tr>
<td>feasopt</td>
<td>computes a minimum-cost relaxation to make an infeasible model feasible</td>
<td>0</td>
</tr>
<tr>
<td>feasoptmode</td>
<td>Mode of FeasOpt</td>
<td>0</td>
</tr>
<tr>
<td>.feaspref</td>
<td>feasibility preference</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>allow interactive option setting after a Control-C</td>
<td>0</td>
</tr>
<tr>
<td>lpmethod</td>
<td>algorithm to be used for LP problems</td>
<td>0</td>
</tr>
<tr>
<td>memoryemphasis</td>
<td>Reduces use of memory</td>
<td>0</td>
</tr>
<tr>
<td>names</td>
<td>load GAMS names into Cplex</td>
<td>1</td>
</tr>
<tr>
<td>numericalemphasis</td>
<td>emphasizes precision in numerically unstable or difficult problems</td>
<td>0</td>
</tr>
<tr>
<td>objrng</td>
<td>do objective ranging</td>
<td>no objective ranging is done</td>
</tr>
<tr>
<td>parallelmode</td>
<td>parallel optimization mode</td>
<td>1</td>
</tr>
<tr>
<td>predual</td>
<td>give dual problem to the optimizer</td>
<td>0</td>
</tr>
<tr>
<td>preind</td>
<td>turn presolver on/off</td>
<td>1</td>
</tr>
<tr>
<td>prelinear</td>
<td>linear reduction indicator</td>
<td>1</td>
</tr>
<tr>
<td>prepass</td>
<td>number of presolve applications to perform</td>
<td>-1</td>
</tr>
<tr>
<td>printoptions</td>
<td>list values of all options to GAMS listing file</td>
<td>0</td>
</tr>
<tr>
<td>qpmethod</td>
<td>algorithm to be used for QP problems</td>
<td>0</td>
</tr>
<tr>
<td>qtolin</td>
<td>linearization of the quadratic terms in the objective function of a QP or MIQP model</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>reduce</td>
<td>primal and dual reduction type</td>
<td>3</td>
</tr>
<tr>
<td>relaxpreind</td>
<td>presolve for initial relaxation on/off</td>
<td>-1</td>
</tr>
<tr>
<td>rerun</td>
<td>rerun problem if presolve infeasible or unbounded</td>
<td>yes</td>
</tr>
<tr>
<td>rhsrng</td>
<td>do right-hand-side ranging</td>
<td>no right-hand-side ranging is done</td>
</tr>
<tr>
<td>rngrestart</td>
<td>write GAMS readable ranging information file</td>
<td>ranging information is printed to the listing file</td>
</tr>
<tr>
<td>scaind</td>
<td>matrix scaling on/off</td>
<td>0</td>
</tr>
<tr>
<td>solutiontarget</td>
<td>type of solution when solving a nonconvex continuous quadratic model</td>
<td>0</td>
</tr>
<tr>
<td>threads</td>
<td>global default thread count</td>
<td>GAMS Threads</td>
</tr>
<tr>
<td>tilim</td>
<td>overrides the GAMS ResLim option</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>tuning</td>
<td>invokes parameter tuning tool</td>
<td></td>
</tr>
<tr>
<td>tuningdettlim</td>
<td>tuning deterministic time limit per model or suite</td>
<td>1e+007</td>
</tr>
<tr>
<td>tuningdisplay</td>
<td>level of information reported by the tuning tool</td>
<td>1</td>
</tr>
<tr>
<td>tuningmeasure</td>
<td>measure for evaluating progress for a suite of models</td>
<td>1</td>
</tr>
<tr>
<td>tuningrepeat</td>
<td>number of times tuning is to be repeated on perturbed versions</td>
<td>1</td>
</tr>
<tr>
<td>tuningtilim</td>
<td>tuning time limit per model or suite</td>
<td>0.2*GAMS ResLim</td>
</tr>
<tr>
<td>workdir</td>
<td>directory for working files</td>
<td>current or project directory</td>
</tr>
<tr>
<td>workmem</td>
<td>memory available for working storage</td>
<td>128</td>
</tr>
</tbody>
</table>

### 5.2 Simplex Algorithmic Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash</td>
<td>crash strategy (used to obtain starting basis)</td>
<td>1</td>
</tr>
<tr>
<td>dpriind</td>
<td>dual simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>epper</td>
<td>perturbation constant</td>
<td>1e-006</td>
</tr>
<tr>
<td>iis</td>
<td>run the IIS finder if the problem is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>netfind</td>
<td>attempt network extraction</td>
<td>2</td>
</tr>
<tr>
<td>netppriind</td>
<td>network simplex pricing</td>
<td>0</td>
</tr>
<tr>
<td>perind</td>
<td>force initial perturbation</td>
<td>0</td>
</tr>
<tr>
<td>perlim</td>
<td>number of stalled iterations before perturbation</td>
<td>0</td>
</tr>
<tr>
<td>ppriind</td>
<td>primal simplex pricing</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.3 Simplex Limit Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>itlim</td>
<td>iteration limit</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>netitlim</td>
<td>iteration limit for network simplex</td>
<td>large</td>
</tr>
<tr>
<td>objlim</td>
<td>objective function lower limit</td>
<td>-1e+075</td>
</tr>
<tr>
<td>objulim</td>
<td>objective function upper limit</td>
<td>1e+075</td>
</tr>
<tr>
<td>singlim</td>
<td>limit on singularity repairs</td>
<td>10</td>
</tr>
</tbody>
</table>

### 5.4 Simplex Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>epmrk</td>
<td>Markowitz pivot tolerance</td>
<td>0.01</td>
</tr>
<tr>
<td>eopt</td>
<td>optimality tolerance</td>
<td>1e-006</td>
</tr>
<tr>
<td>eprhs</td>
<td>feasibility tolerance</td>
<td>1e-006</td>
</tr>
<tr>
<td>neteopt</td>
<td>optimality tolerance for the network simplex method</td>
<td>1e-006</td>
</tr>
<tr>
<td>neteprhs</td>
<td>feasibility tolerance for the network simplex method</td>
<td>1e-006</td>
</tr>
</tbody>
</table>

### 5.5 Barrier Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>baralg</td>
<td>algorithm selection</td>
<td>0</td>
</tr>
<tr>
<td>barcolnz</td>
<td>dense column handling</td>
<td>0</td>
</tr>
<tr>
<td>barcrossalg</td>
<td>barrier crossover method</td>
<td>0</td>
</tr>
<tr>
<td>barepcomp</td>
<td>convergence tolerance</td>
<td>1e-008</td>
</tr>
<tr>
<td>bargrowth</td>
<td>unbounded face detection</td>
<td>1e+012</td>
</tr>
<tr>
<td>baritlim</td>
<td>iteration limit</td>
<td>large</td>
</tr>
<tr>
<td>barmaxcor</td>
<td>maximum correction limit</td>
<td>-1</td>
</tr>
<tr>
<td>barobjrng</td>
<td>maximum objective function</td>
<td>1e+020</td>
</tr>
<tr>
<td>barorder</td>
<td>row ordering algorithm selection</td>
<td>0</td>
</tr>
<tr>
<td>barmqcepcenp</td>
<td>convergence tolerance for the barrier optimizer for QCPs</td>
<td>1e-007</td>
</tr>
<tr>
<td>barstartalg</td>
<td>barrier starting point algorithm</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.6 Sifting Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>sifalg</td>
<td>sifting subproblem algorithm</td>
<td>0</td>
</tr>
<tr>
<td>sifitlim</td>
<td>limit on sifting iterations</td>
<td>large</td>
</tr>
</tbody>
</table>
### 5.7 MIP Algorithmic Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbinterval</td>
<td>best bound interval</td>
<td>7</td>
</tr>
<tr>
<td>bndstrenind</td>
<td>bound strengthening</td>
<td>-1</td>
</tr>
<tr>
<td>brdir</td>
<td>set branching direction</td>
<td>0</td>
</tr>
<tr>
<td>bttol</td>
<td>backtracking limit</td>
<td>0.9999</td>
</tr>
<tr>
<td>cliques</td>
<td>clique cut generation</td>
<td>0</td>
</tr>
<tr>
<td>covers</td>
<td>cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>cutlo</td>
<td>lower cutoff for tree search</td>
<td>-1e+075</td>
</tr>
<tr>
<td>cuts</td>
<td>default cut generation</td>
<td>0</td>
</tr>
<tr>
<td>cutsfactor</td>
<td>cut limit</td>
<td>4</td>
</tr>
<tr>
<td>cutup</td>
<td>upper cutoff for tree search</td>
<td>1e+075</td>
</tr>
<tr>
<td>disjcuts</td>
<td>disjunctive cuts generation</td>
<td>0</td>
</tr>
<tr>
<td>divetype</td>
<td>MIP dive strategy</td>
<td>0</td>
</tr>
<tr>
<td>eachcutlim</td>
<td>Sets a limit for each type of cut</td>
<td>2100000000</td>
</tr>
<tr>
<td>flowcovers</td>
<td>flow cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>flowpaths</td>
<td>flow path cut generation</td>
<td>0</td>
</tr>
<tr>
<td>fpheur</td>
<td>feasibility pump heuristic</td>
<td>0</td>
</tr>
<tr>
<td>fraccuts</td>
<td>Gomory fractional cut generation</td>
<td>0</td>
</tr>
<tr>
<td>gubcovers</td>
<td>GUB cover cut generation</td>
<td>0</td>
</tr>
<tr>
<td>heurfreq</td>
<td>heuristic frequency</td>
<td>0</td>
</tr>
<tr>
<td>implbd</td>
<td>implied bound cut generation</td>
<td>0</td>
</tr>
<tr>
<td>lbheur</td>
<td>local branching heuristic</td>
<td>0</td>
</tr>
<tr>
<td>liftprojcuts</td>
<td>lift-and-project cuts</td>
<td>0</td>
</tr>
<tr>
<td>localimplied</td>
<td>generation of locally valid implied bound cuts</td>
<td>0</td>
</tr>
<tr>
<td>mfcuts</td>
<td>multi-commodity flow cut generation</td>
<td>0</td>
</tr>
<tr>
<td>mipemphasis</td>
<td>MIP solution tactics</td>
<td>0</td>
</tr>
<tr>
<td>mipkappastats</td>
<td>MIP kappa computation</td>
<td>-1</td>
</tr>
<tr>
<td>mipordind</td>
<td>priority list on/off</td>
<td>GAMS PriorOpt</td>
</tr>
<tr>
<td>mipordtype</td>
<td>priority order generation</td>
<td>0</td>
</tr>
<tr>
<td>mipsearch</td>
<td>search strategy for mixed integer programs</td>
<td>0</td>
</tr>
<tr>
<td>mipstart</td>
<td>use mip starting values</td>
<td>0</td>
</tr>
<tr>
<td>miqcpstrat</td>
<td>MIQCP relaxation choice</td>
<td>0</td>
</tr>
<tr>
<td>mircuts</td>
<td>mixed integer rounding cut generation</td>
<td>0</td>
</tr>
<tr>
<td>nodefileind</td>
<td>node storage file indicator</td>
<td>1</td>
</tr>
<tr>
<td>nodesel</td>
<td>node selection strategy</td>
<td>1</td>
</tr>
<tr>
<td>preslvnd</td>
<td>node presolve selector</td>
<td>0</td>
</tr>
<tr>
<td>probe</td>
<td>perform probing before solving a MIP</td>
<td>0</td>
</tr>
<tr>
<td>qpmakepsdind</td>
<td>adjust MIQP formulation to make the quadratic matrix positive-semi-definite</td>
<td>1</td>
</tr>
<tr>
<td>relaxfixedinfeas</td>
<td>access small infeasibilities in the solve of the fixed problem</td>
<td>0</td>
</tr>
<tr>
<td>repeatpresolve</td>
<td>reapply presolve at root after preprocessing</td>
<td>-1</td>
</tr>
<tr>
<td>rinsheur</td>
<td>relaxation induced neighborhood search frequency</td>
<td>0</td>
</tr>
<tr>
<td>solvefinal</td>
<td>switch to solve the problem with fixed discrete variables</td>
<td>1</td>
</tr>
<tr>
<td>startalg</td>
<td>MIP starting algorithm</td>
<td>0</td>
</tr>
</tbody>
</table>
### 5.8 MIP Limit Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongcandlim</td>
<td>size of the candidates list for strong branching</td>
<td>10</td>
</tr>
<tr>
<td>strongitlim</td>
<td>limit on iterations per branch for strong branching</td>
<td>0</td>
</tr>
<tr>
<td>subalg</td>
<td>algorithm for subproblems</td>
<td>0</td>
</tr>
<tr>
<td>submipnodelim</td>
<td>limit on number of nodes in an RINS subMIP</td>
<td>500</td>
</tr>
<tr>
<td>symmetry</td>
<td>symmetry breaking cuts</td>
<td>-1</td>
</tr>
<tr>
<td>varsel</td>
<td>variable selection strategy at each node</td>
<td>0</td>
</tr>
<tr>
<td>zerohalfcuts</td>
<td>zero-half cuts</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.9 MIP Solution Pool Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>.divflt</td>
<td>solution pool range filter coefficients</td>
<td>0</td>
</tr>
<tr>
<td>divfltlo</td>
<td>lower bound on diversity</td>
<td>mindouble</td>
</tr>
<tr>
<td>divftup</td>
<td>upper bound on diversity</td>
<td>maxdouble</td>
</tr>
<tr>
<td>populatelim</td>
<td>limit of solutions generated for the solution pool by populate method</td>
<td>20</td>
</tr>
<tr>
<td>randomseed</td>
<td>sets the random seed differently for diversity of solutions</td>
<td>changes with each Cplex release</td>
</tr>
<tr>
<td>readflt</td>
<td>reads Cplex solution pool filter file</td>
<td></td>
</tr>
<tr>
<td>solnpool</td>
<td>solution pool file name</td>
<td></td>
</tr>
<tr>
<td>solnpoolagap</td>
<td>absolute tolerance for the solutions in the solution pool</td>
<td>1e+075</td>
</tr>
<tr>
<td>solnpoolcapacity</td>
<td>limits of solutions kept in the solution pool</td>
<td>2100000000</td>
</tr>
<tr>
<td>solnpoolgap</td>
<td>relative tolerance for the solutions in the solution pool</td>
<td>1e+075</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>solnpoolintensity</td>
<td>solution pool intensity for ability to produce multiple solutions</td>
<td>0</td>
</tr>
<tr>
<td>solnpoolmerge</td>
<td>solution pool file name for merged solutions</td>
<td></td>
</tr>
<tr>
<td>solnpoolnumsym</td>
<td>maximum number of variable symbols when writing merged solutions</td>
<td>10</td>
</tr>
<tr>
<td>solnpoolpop</td>
<td>methods to populate the solution pool</td>
<td>1</td>
</tr>
<tr>
<td>solnpoolpopdel</td>
<td>file with solution numbers to delete from the solution pool</td>
<td></td>
</tr>
<tr>
<td>solnpoolpoprepeat</td>
<td>method to decide if populating the solution should be repeated</td>
<td></td>
</tr>
<tr>
<td>solnpoolprefix</td>
<td>file name prefix for GDX solution files</td>
<td>soln</td>
</tr>
<tr>
<td>solnpoolreplace</td>
<td>strategy for replacing a solution in the solution pool</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.10 MIP Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>epagap</td>
<td>absolute stopping tolerance</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td>epgap</td>
<td>relative stopping tolerance</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>epint</td>
<td>integrality tolerance</td>
<td>1e-005</td>
</tr>
<tr>
<td>objdif</td>
<td>overrides GAMS Cheat parameter</td>
<td>0</td>
</tr>
<tr>
<td>relobjdif</td>
<td>relative cheat parameter</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.11 Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bardisplay</td>
<td>progress display level</td>
<td>1</td>
</tr>
<tr>
<td>clonelog</td>
<td>enable clone logs</td>
<td>0</td>
</tr>
<tr>
<td>mipdisplay</td>
<td>progress display level</td>
<td>4</td>
</tr>
<tr>
<td>mipinterval</td>
<td>progress display interval</td>
<td>0</td>
</tr>
<tr>
<td>mpslongnum</td>
<td>MPS file format precision of numeric output</td>
<td>1</td>
</tr>
<tr>
<td>netdisplay</td>
<td>network display level</td>
<td>2</td>
</tr>
<tr>
<td>quality</td>
<td>write solution quality statistics</td>
<td>0</td>
</tr>
<tr>
<td>siftdisplay</td>
<td>sifting display level</td>
<td>1</td>
</tr>
<tr>
<td>simdisplay</td>
<td>simplex display level</td>
<td>1</td>
</tr>
<tr>
<td>writebas</td>
<td>produce a Cplex basis file</td>
<td></td>
</tr>
<tr>
<td>writeflt</td>
<td>produce a Cplex solution pool filter file</td>
<td></td>
</tr>
<tr>
<td>writelp</td>
<td>produce a Cplex LP file</td>
<td></td>
</tr>
<tr>
<td>writemps</td>
<td>produce a Cplex MPS file</td>
<td></td>
</tr>
<tr>
<td>writemst</td>
<td>produce a Cplex mst file</td>
<td></td>
</tr>
<tr>
<td>writeord</td>
<td>produce a Cplex ord file</td>
<td></td>
</tr>
<tr>
<td>writeparam</td>
<td>produce a Cplex parameter file with all active options</td>
<td></td>
</tr>
<tr>
<td>writepre</td>
<td>produce a Cplex LP/MPS/SAV file of the presolved problem</td>
<td></td>
</tr>
<tr>
<td>writesav</td>
<td>produce a Cplex binary problem file</td>
<td></td>
</tr>
</tbody>
</table>
5.12 BCH Facility Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userincbcall</td>
<td>The GAMS command line to call the incumbent checking program</td>
</tr>
</tbody>
</table>

5.13 The GAMS/Cplex Options File

The GAMS/Cplex options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file 'cplex.opt'.

```
scaind 1
simdisplay 2
```

It will cause Cplex to use a more aggressive scaling method than the default. The iteration log will have an entry for each iteration instead of an entry for each refactorization.

6 Special Notes

6.1 Physical Memory Limitations

For the sake of computational speed, Cplex should use only available physical memory rather than virtual or paged memory. When Cplex recognizes that a limited amount of memory is available it automatically makes algorithmic adjustments to compensate. These adjustments almost always reduce optimization speed. Learning to recognize when these automatic adjustments occur can help to determine when additional memory should be added to the computer.

On virtual memory systems, if memory paging to disk is observed, a considerable performance penalty is incurred. Increasing available memory will speed the solution process dramatically. Also consider option memoryemphasis to conserve memory where possible.

Cplex performs an operation called refactorization at a frequency determined by the reinv option setting. The longer Cplex works between refactorizations, the greater the amount of memory required to complete each iteration. Therefore, one means for conserving memory is to increase the refactorization frequency. Since refactorizing is an expensive operation, increasing the refactorization frequency by reducing the reinv option setting generally will slow performance. Cplex will automatically increase the refactorization frequency if it encounters low memory availability. This can be seen by watching the iteration log. The default log reports problem status at every refactorization. If the number of iterations between iteration log entries is decreasing, Cplex is increasing the refactorization frequency. Since Cplex might increase the frequency to once per iteration, the impact on performance can be dramatic. Providing additional memory should be beneficial.

6.2 Using Special Ordered Sets

For some models a special structure can be exploited. GAMS allows you to declare SOS1 and SOS2 variables (Special Ordered Sets of type 1 and 2).

In Cplex the definition for SOS1 variables is:

- A set of variables for which at most one variable may be non-zero.

The definition for SOS2 variables is:

- A set of variables for which at most two variables may be non-zero. If two variables are non-zero, they must be adjacent in the set.
6.3 Using Semi-Continuous and Semi-Integer Variables

GAMS allows the declaration of semi-continuous and semi-integer variables. These variable types are directly supported by GAMS/Cplex. For example:

```gams
SemiCont Variable x;
x.lo = 3.2;
x.up = 8.7;

SemiInt Variable y;
y.lo = 5;
y.up = 10;
```

Variable x will be allowed to take on a value of 0.0 or any value between 3.2 and 8.7. Variable y will be allowed to take on a value of 0 or any integral value between 5 and 10.

Note that Cplex requires a finite upper bound for semi-continuous and semi-integer variables.

6.4 Running Out of Memory for MIP Problems

The most common difficulty when solving MIP problems is running out of memory. This problem arises when the branch and bound tree becomes so large that insufficient memory is available to solve an LP subproblem. As memory gets tight, you may observe frequent warning messages while Cplex attempts to navigate through various operations within limited memory. If a solution is not found shortly the solution process will be terminated with an unrecoverable integer failure message.

The tree information saved in memory can be substantial. Cplex saves a basis for every unexplored node. When utilizing the best bound method of node selection, the list of such nodes can become very long for large or difficult problems. How large the unexplored node list can become is entirely dependent on the actual amount of physical memory available and the actual size of the problem. Certainly increasing the amount of memory available extends the problem solving capability. Unfortunately, once a problem has failed because of insufficient memory, you can neither project how much further the process needed to go nor how much memory would be required to ultimately solve it.

Memory requirements can be limited by using the workmem, option with the nodefileind option. Setting nodefileind to 2 or 3 will cause Cplex to store portions of the branch and bound tree on disk whenever it grows to larger than the size specified by option workmem. That size should be set to something less than the amount of physical memory available.

Another approach is to modify the solution process to utilize less memory.

- Set option nodesel to use a best estimate strategy or, more drastically a depth-first-search. Depth first search rarely generates a large unexplored node list since Cplex will be diving deep into the branch and bound tree rather than jumping around within it.
- Set option varsel to use strong branching. Strong branching spends extra computation time at each node to choose a better branching variable. As a result it generates a smaller tree. It is often faster overall, as well.
- On some problems, a large number of cuts will be generated without a correspondingly large benefit in solution speed. Cut generation can be turned off using option cuts.

6.5 Failing to Prove Integer Optimality

One frustrating aspect of the branch and bound technique for solving MIP problems is that the solution process can continue long after the best solution has been found. Remember that the branch and bound tree may be as large as $2^n$ nodes, where $n$ equals the number of binary variables. A problem containing only 30 binary variables could produce a tree having over one billion nodes! If no other stopping criteria have been set, the process might continue ad infinitum until the search is complete or your computer’s memory is exhausted.

In general you should set at least one limit on the optimization process before beginning an optimization. Setting limits ensures that an exhaustive tree search will terminate in reasonable time. Once terminated, you can rerun the problem using
some different option settings. Consider some of the shortcuts described previously for improving performance including setting the options for mip gap, objective value difference, upper cutoff, or lower cutoff.

6.6 Starting from a MIP Solution

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution. When you provide such a starting solution, you may invoke relaxation induced neighborhood search (RINS heuristic) or solution polishing to improve the given solution. This first integer solution may include continuous and discrete variables of various types, such as semi-continuous variables or special ordered sets.

If you specify values for all discrete variables, GAMS/CPLEX will check the validity of the values as an integer-feasible solution; if you specify values for only a portion of the discrete variables, GAMS/CPLEX will attempt to fill in the missing values in a way that leads to an integer-feasible solution. If the specified values do not lead directly to an integer-feasible solution, GAMS/CPLEX will apply a quick heuristic to try to repair the MIP Start. The number of times that GAMS/CPLEX applies the heuristic is controlled by the repair tries parameter (RepairTries). If this process succeeds, the solution will be treated as an integer solution of the current problem.

A MIP start will only be used by GAMS/CPLEX if the MipStart parameter is set to 1.

6.7 Using the Feasibility Relaxation

The feasibility relaxation is enabled by the FeasOpt parameter in a CPLEX solver option file.

With the FeasOpt option CPLEX accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the .feaspref option. A negative or zero preference means that the associated bound or constraint is not to be modified. The weighted penalty function is constructed from these preferences. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. However, it is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a CPLEX solver option file. The syntax is:

(variable or equation) .feaspref (value)

For example, suppose we have a GAMS declaration:

```gams
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the cplex.opt file can be specified by:

```plaintext
feasopt 1
v.feaspref 1
v.feaspref('i1','*) 2
v.feaspref('i1','j2') 0
e.feaspref('*,j1') 0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Futhermore, we specify that all variables v(i,j) have preference of 1, except variables over set element i1, which have a preference of 2. The variable over set element i1 and j2 has preference~0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The
same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords variables or equations instead of the individual variable and equations names (e.g. variables.feaspref 1).

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations). Please check description of parameter FeasOpt FeasOptMode for details. Also check example models feasopt∗ in the GAMS Model library.

7 GAMS/Cplex Log File

Cplex reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the primal simplex algorithm, the iteration log starts with the iteration number followed by the scaled infeasibility value. Once feasibility has been attained, the objective function value is listed instead. At the default value for option simdisplay there is a log line for each refactorization. The screen log has the following appearance:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time = 0.01 sec.
Using conservative initial basis.
Iteration log . . .
Iteration:  1  Scaled infeas =  193998.067174
Iteration: 29  Objective =  -3484.286415
Switched to devex.
Iteration:  98  Objective =  -1852.931117
Iteration: 166  Objective =  -349.706562

Optimal solution found.

Objective :  901.161538
```

The iteration log for the dual simplex algorithm is similar, but the dual infeasibility and dual objective are reported instead of the corresponding primal values:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time = 0.01 sec.
Iteration log . . .
Iteration:  1  Scaled dual infeas =  3.890823
Iteration: 53  Dual objective =  4844.392441
Iteration: 114  Dual objective =  1794.360714
Iteration: 176  Dual objective =  1120.183325
```

```
Iteration: 238  Dual objective  =  915.143030
Removing shift (1).

Optimal solution found.

Objective :  901.161538

The log for the network algorithm adds statistics about the extracted network and a log of the network iterations. The optimization is finished by one of the simplex algorithms and an iteration log for that is produced as well.

Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time = 0.01 sec.
Extracted network with 25 nodes and 116 arcs.
Extraction time = -0.00 sec.
Iteration log . . .
Iteration: 0  Infeasibility  =  1232.378800 (-1.32326e+12)

Network - Optimal: Objective =  1.5716820779e+03
Network time = 0.01 sec.  Iterations = 26 (24)

Iteration log . . .
Iteration: 1  Scaled infeas  =  212696.154729
Iteration: 62  Scaled infeas  =  10020.401232
Iteration: 142  Scaled infeas  =  4985.200129
Switched to devex.
Iteration: 217  Objective  =  -3883.782587
Iteration: 291  Objective  =  -1423.126582

Optimal solution found.

Objective :  901.161538

The log for the barrier algorithm adds various algorithm specific statistics about the problem before starting the iteration log. The iteration log includes columns for primal and dual objective values and infeasibility values. A special log follows for the crossover to a basic solution.

Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time = 0.02 sec.
Number of nonzeros in lower triangle of A*A' = 6545
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.01 sec.
Summary statistics for Cholesky factor:

<table>
<thead>
<tr>
<th>Rows in Factor</th>
<th>Integer space required</th>
<th>Total non-zeros in factor</th>
<th>Total FP ops to factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>243</td>
<td>578</td>
<td>8491</td>
<td>410889</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itn</th>
<th>Primal Obj</th>
<th>Dual Obj</th>
<th>Prim Inf</th>
<th>Upper Inf</th>
<th>Dual Inf</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1.2826603e+06</td>
<td>7.4700787e+08</td>
<td>2.25e+10</td>
<td>6.13e+06</td>
<td>4.00e+05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>-2.6426196e+05</td>
<td>6.352653e+08</td>
<td>4.58e+09</td>
<td>1.25e+06</td>
<td>1.35e+05</td>
</tr>
<tr>
<td>2</td>
<td>-9.917854e+04</td>
<td>4.1669756e+08</td>
<td>1.66e+09</td>
<td>4.52e+05</td>
<td>3.93e+04</td>
</tr>
<tr>
<td>3</td>
<td>-2.6624468e+04</td>
<td>2.1507018e+08</td>
<td>3.80e+08</td>
<td>1.04e+05</td>
<td>1.35e+05</td>
</tr>
<tr>
<td>4</td>
<td>-1.2104334e+04</td>
<td>7.8523364e+07</td>
<td>9.69e+07</td>
<td>2.65e+04</td>
<td>2.52e+03</td>
</tr>
<tr>
<td>5</td>
<td>-9.5217661e+03</td>
<td>3.619431e+07</td>
<td>3.13e-07</td>
<td>6.84e-12</td>
<td>9.92e+02</td>
</tr>
<tr>
<td>6</td>
<td>-8.6929410e+03</td>
<td>1.4134077e+07</td>
<td>4.94e-07</td>
<td>1.35e+03</td>
<td>1.35e+05</td>
</tr>
<tr>
<td>7</td>
<td>-8.3726267e+03</td>
<td>3.1985844e+07</td>
<td>4.94e-07</td>
<td>1.35e+03</td>
<td>1.35e+05</td>
</tr>
<tr>
<td>8</td>
<td>-3.8181279e+03</td>
<td>2.616059e+07</td>
<td>3.80e+08</td>
<td>9.37e-12</td>
<td>2.50e-02</td>
</tr>
<tr>
<td>9</td>
<td>-5.1366439e+03</td>
<td>2.8102021e+07</td>
<td>5.91e-07</td>
<td>1.91e-11</td>
<td>3.00e-04</td>
</tr>
<tr>
<td>10</td>
<td>-1.9771576e+03</td>
<td>1.5960442e+07</td>
<td>3.43e-06</td>
<td>7.02e-12</td>
<td>3.81e-03</td>
</tr>
<tr>
<td>11</td>
<td>-4.346261e+02</td>
<td>8.3443795e+07</td>
<td>4.99e-07</td>
<td>1.22e-11</td>
<td>7.93e-04</td>
</tr>
<tr>
<td>12</td>
<td>1.2882968e+02</td>
<td>5.2138155e+07</td>
<td>2.22e-07</td>
<td>1.45e-11</td>
<td>8.72e-04</td>
</tr>
<tr>
<td>13</td>
<td>5.0418542e+02</td>
<td>5.3678066e+07</td>
<td>1.45e-07</td>
<td>1.26e-11</td>
<td>7.93e-04</td>
</tr>
<tr>
<td>14</td>
<td>2.4951043e+02</td>
<td>6.5911879e+07</td>
<td>1.73e-07</td>
<td>1.43e-11</td>
<td>5.33e-04</td>
</tr>
<tr>
<td>15</td>
<td>2.4666057e+02</td>
<td>7.6179064e+07</td>
<td>7.83e-06</td>
<td>2.17e-11</td>
<td>3.15e-04</td>
</tr>
<tr>
<td>16</td>
<td>4.6820025e+02</td>
<td>8.1319322e+07</td>
<td>4.75e-06</td>
<td>1.78e-11</td>
<td>2.57e-04</td>
</tr>
<tr>
<td>17</td>
<td>5.6081604e+02</td>
<td>7.9608915e+07</td>
<td>3.09e-06</td>
<td>9.18e-11</td>
<td>2.89e-04</td>
</tr>
<tr>
<td>18</td>
<td>6.4517294e+02</td>
<td>7.7729659e+07</td>
<td>1.61e-06</td>
<td>1.27e-11</td>
<td>3.29e-04</td>
</tr>
<tr>
<td>19</td>
<td>7.9603053e+02</td>
<td>7.8584631e+07</td>
<td>5.91e-07</td>
<td>1.91e-11</td>
<td>3.00e-04</td>
</tr>
<tr>
<td>20</td>
<td>8.5871436e+02</td>
<td>8.0198336e+07</td>
<td>1.32e-07</td>
<td>1.46e-11</td>
<td>2.57e-04</td>
</tr>
<tr>
<td>21</td>
<td>8.8146686e+02</td>
<td>8.1244367e+07</td>
<td>1.46e-07</td>
<td>1.84e-11</td>
<td>2.29e-04</td>
</tr>
<tr>
<td>22</td>
<td>8.8327998e+02</td>
<td>8.3544569e+07</td>
<td>1.44e-07</td>
<td>1.96e-11</td>
<td>1.71e-04</td>
</tr>
<tr>
<td>23</td>
<td>8.8595062e+02</td>
<td>8.4926550e+07</td>
<td>1.30e-07</td>
<td>2.85e-11</td>
<td>1.35e-04</td>
</tr>
<tr>
<td>24</td>
<td>8.9780858e+02</td>
<td>8.6318712e+07</td>
<td>1.60e-07</td>
<td>1.08e-11</td>
<td>9.89e-05</td>
</tr>
<tr>
<td>25</td>
<td>8.9940069e+02</td>
<td>8.9108502e+07</td>
<td>1.78e-07</td>
<td>1.07e-11</td>
<td>2.62e-05</td>
</tr>
<tr>
<td>26</td>
<td>8.9979049e+02</td>
<td>8.9138752e+07</td>
<td>5.14e-07</td>
<td>1.88e-11</td>
<td>2.54e-05</td>
</tr>
<tr>
<td>27</td>
<td>8.9979401e+02</td>
<td>8.9139850e+02</td>
<td>5.13e-07</td>
<td>2.18e-11</td>
<td>2.54e-05</td>
</tr>
<tr>
<td>28</td>
<td>9.0067378e+02</td>
<td>8.9385969e+02</td>
<td>2.45e-07</td>
<td>1.46e-11</td>
<td>1.90e-05</td>
</tr>
<tr>
<td>29</td>
<td>9.0112149e+02</td>
<td>8.9746581e+02</td>
<td>2.12e-07</td>
<td>1.71e-11</td>
<td>9.61e-06</td>
</tr>
<tr>
<td>30</td>
<td>9.0113610e+02</td>
<td>8.9837069e+02</td>
<td>2.11e-07</td>
<td>1.31e-11</td>
<td>7.40e-06</td>
</tr>
<tr>
<td>31</td>
<td>9.0116361e+02</td>
<td>8.9982723e+02</td>
<td>1.90e-07</td>
<td>2.12e-11</td>
<td>3.53e-06</td>
</tr>
<tr>
<td>32</td>
<td>9.0116564e+02</td>
<td>9.0088083e+02</td>
<td>2.92e-07</td>
<td>1.27e-11</td>
<td>7.35e-07</td>
</tr>
<tr>
<td>33</td>
<td>9.0116131e+02</td>
<td>9.0116262e+02</td>
<td>3.07e-07</td>
<td>1.81e-11</td>
<td>3.13e-09</td>
</tr>
<tr>
<td>34</td>
<td>9.0116154e+02</td>
<td>9.0116154e+02</td>
<td>4.85e-07</td>
<td>1.69e-11</td>
<td>9.72e-13</td>
</tr>
</tbody>
</table>

Barrier time = 0.39 sec.

Primal crossover.
Primal: Fixing 13 variables.
12 PMoves: Infeasibility 1.97677059e-06 Objective 9.01161542e+02
0 PMoves: Infeasibility 0.00000000e+00 Objective 9.01161540e+02
Primal: Pushed 1, exchanged 12.
Dual: Fixing 3 variables.
2 DMoves: Infeasibility 1.28422758e-36 Objective 9.01161540e+02
0 DMoves: Infeasibility 1.28422758e-36 Objective 9.01161540e+02
Dual: Pushed 3, exchanged 0.
Using devex.
Total crossover time = 0.02 sec.

Optimal solution found.

Objective : 901.161540

For MIP problems, during the branch and bound search, Cplex reports the node number, the number of nodes left, the value of the Objective function, the number of integer variables that have fractional values, the current best integer solution, the best relaxed solution at a node and an iteration count. The last column show the current optimality gap as a percentage. CPLEX
logs an asterisk (*) in the left-most column for any node where it finds an integer-feasible solution or new incumbent. The + denotes an incumbent generated by the heuristic.

Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
Reduced MIP has 99 rows, 76 columns, and 419 nonzeros.
Presolve time = 0.00 sec.

Iteration log . . .
Iteration: 1 Dual objective = 0.000000
Root relaxation solution time = 0.01 sec.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cuts/</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Left</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>0+</td>
</tr>
<tr>
<td>*</td>
<td>50+</td>
</tr>
<tr>
<td>100</td>
<td>99</td>
</tr>
</tbody>
</table>

Fixing integer variables, and solving final LP. . .
Tried aggregator 1 time.
LP Presolve eliminated 100 rows and 77 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec.

Solution satisfies tolerances.

MIP Solution : 4.000000 (2650 iterations, 185 nodes)
Final LP : 4.000000 (0 iterations)

Best integer solution possible : 1.000000
Absolute gap : 3
Relative gap : 1.5

8 Detailed Descriptions of CPLEX Options

These options should be entered in the options file after setting the GAMS ModelName.OptFile parameter to 1. The name of the options file is cplex.opt. The options file is case insensitive and the keywords should be given in full.

**advind** (integer): advanced basis use

Use an Advanced Basis. GAMS/Cplex will automatically use an advanced basis from a previous solve statement. The GAMS Bratio option can be used to specify when not to use an advanced basis. The Cplex option advind can be used to ignore a basis passed on by GAMS (it overrides Bratio).

(default = determined by GAMS Bratio)

0 Do not use advanced basis
1 Use advanced basis if available
2 Crash an advanced basis if available (use basis with presolve)

**aggcutlim** (integer): aggregation limit for cut generation

Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts. For most purposes, the default will be satisfactory.

(default = 3)
**aggfill (integer):** aggregator fill parameter

Aggregator fill limit. If the net result of a single substitution is more non-zeros than the setting of the `aggfill` parameter, the substitution will not be made.

(default = 10)

**aggind (integer):** aggregator on/off

This option, when set to a nonzero value, will cause the Cplex aggregator to use substitution where possible to reduce the number of rows and columns in the problem. If set to a positive value, the aggregator will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, the aggregator is applied once for linear programs and an unlimited number of times for mixed integer problems.

(default = -1)

-1 Once for LP, unlimited for MIP
0 Do not use

**auxrootthreads (integer):** number of threads for auxiliary tasks at the root node

Partitions the number of threads for CPLEX to use for auxiliary tasks while it solves the root node of a problem. On a system that offers \( N \) processors or \( N \) global threads, if you set this parameter to \( n \), where \( N>n>0 \) then CPLEX uses at most \( n \) threads for auxiliary tasks and at most \( N-n \) threads to solve the root node. See also the parameter `threads`.

You cannot set \( n \), the value of this parameter, to a value greater than or equal to \( N \), the number of processors or global threads offered on your system. In other words, when you set this parameter to a value other than its default, that value must be strictly less than the number of processors or global threads on your system. Independent of the auxiliary root threads parameter, CPLEX will never use more threads than those defined by the global default thread count parameter. CPLEX also makes sure that there is at least one thread available for the main root tasks. For example, if you set the global threads parameter to 3 and the auxiliary root threads parameter to 4, CPLEX still uses only two threads for auxiliary root tasks in order to keep one thread available for the main root tasks. At its default value, 0 (zero), CPLEX automatically chooses the number of threads to use for the primary root tasks and for auxiliary tasks. The number of threads that CPLEX uses to solve the root node depends on several factors: 1) the number of processors available on your system; 2) the number of threads available to your application on your system (for example, as a result of limited resources or competition with other applications); 3) the value of the global default thread count parameter `threads`.

(default = 0)

-1 Off: do not use additional threads for auxiliary tasks
0 Automatic: let CPLEX choose the number of threads to use
\( n>n>0 \) Use \( n \) threads for auxiliary root tasks

**baralg (integer):** algorithm selection

Selects which barrier algorithm to use. The default setting of 0 uses the infeasibility-estimate start algorithm for MIP subproblems and the standard barrier algorithm, option 3, for other cases. The standard barrier algorithm is almost always fastest. The alternative algorithms, options 1 and 2, may eliminate numerical difficulties related to infeasibility, but will generally be slower.

(default = 0)

0 Same as 1 for MIP subproblems, 3 otherwise
1 Infeasibility-estimate start
2 Infeasibility-constant start
3 standard barrier algorithm

**barcolnz (integer):** dense column handling

Determines whether or not columns are considered dense for special barrier algorithm handling. At the default setting of 0, this parameter is determined dynamically. Values above 0 specify the number of entries in columns
to be considered as dense.
(default = 0)

barcrossalg (integer): barrier crossover method
Selects which, if any, crossover method is used at the end of a barrier optimization.
(default = 0)
-1 No crossover
0 Automatic
1 Primal crossover
2 Dual crossover

bardisplay (integer): progress display level
Determines the level of progress information to be displayed while the barrier method is running.
(default = 1)
0 No progress information
1 Display normal information
2 Display diagnostic information

barepcomp (real): convergence tolerance
Determines the tolerance on complementarity for convergence of the barrier algorithm. The algorithm will
terminate with an optimal solution if the relative complementarity is smaller than this value.
(default = 1e-008)

bargrowth (real): unbounded face detection
Used by the barrier algorithm to detect unbounded optimal faces. At higher values, the barrier algorithm will
be less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical
difficulties if the problem does have an unbounded face.
(default = 1e+012)

baritlim (integer): iteration limit
Determines the maximum number of iterations for the barrier algorithm. When set to 0, no Barrier iterations
occur, but problem setup occurs and information about the setup is displayed (such as Cholesky factorization
information). When left at the default value, there is no explicit limit on the number of iterations.
(default = large)

barmaxcor (integer): maximum correction limit
Specifies the maximum number of centering corrections that should be done on each iteration. Larger values
may improve the numerical performance of the barrier algorithm at the expense of computation time. The default
of -1 means the number is automatically determined.
(default = -1)

barobjrng (real): maximum objective function
Determines the maximum absolute value of the objective function. The barrier algorithm looks at this limit to
detect unbounded problems.
(default = 1e+020)

barorder (integer): row ordering algorithm selection
Determines the ordering algorithm to be used by the barrier method. By default, Cplex attempts to choose the
most effective of the available alternatives. Higher numbers tend to favor better orderings at the expense of
longer ordering runtimes.
(default = 0)
0 Automatic
1 Approximate Minimum Degree (AMD)
2 Approximate Minimum Fill (AMF)
3 Nested Dissection (ND)

**barqcpepcomp (real):** convergence tolerance for the barrier optimizer for QCPs

- **Range:** \([1e^{-012}, 1e+075]\)
- **(default = 1e^{-007})**

**barstartalg (integer):** barrier starting point algorithm

- This option sets the algorithm to be used to compute the initial starting point for the barrier solver. The default starting point is satisfactory for most problems. Since the default starting point is tuned for primal problems, using the other starting points may be worthwhile in conjunction with the predual parameter.
- **(default = 1)**
  - 1 default primal, dual is 0
  - 2 default primal, estimate dual
  - 3 primal average, dual is 0
  - 4 primal average, estimate dual

**bbinterval (integer):** best bound interval

- Set interval for selecting a best bound node when doing a best estimate search. Active only when nodesel is 2 (best estimate). Decreasing this interval may be useful when best estimate is finding good solutions but making little progress in moving the bound. Increasing this interval may help when the best estimate node selection is not finding any good integer solutions. Setting the interval to 1 is equivalent to setting nodesel to 1.
- **(default = 7)**

**bndstrenind (integer):** bound strengthening

- Use bound strengthening when solving mixed integer problems. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during the branch and bound algorithm. This reduction is usually beneficial, but occasionally, due to its iterative nature, takes a long time.
- **(default = -1)**
  - -1 Determine automatically
  - 0 Don’t use bound strengthening
  - 1 Use bound strengthening

**brdir (integer):** set branching direction

- Used to decide which branch (up or down) should be taken first at each node.
- **(default = 0)**
  - -1 Down branch selected first
  - 0 Algorithm decides
  - 1 Up branch selected first

**bttol (real):** backtracking limit

- This option controls how often backtracking is done during the branching process. At each node, Cplex compares the objective function value or estimated integer objective value to these values at parent nodes; the value of the bttol parameter dictates how much relative degradation is tolerated before backtracking. Lower values tend to increase the amount of backtracking, making the search more of a pure best-bound search. Higher values tend to decrease the amount of backtracking, making the search more of a depth-first search. This parameter is used...
only once a first integer solution is found or when a cutoff has been specified.
Range: [0, 1]
(default = 0.9999)

calcqcpduals (integer): calculate the dual values of a quadratically constrained problem
(default = 1)
0 Do not calculate dual values
1 Calculate dual values as long as it does not interfere with presolve reductions
2 Calculate dual values and disable any presolve reductions that would interfere

cliques (integer): clique cut generation
Determines whether or not clique cuts should be generated during optimization.
(default = 0)
~1 Do not generate clique cuts
0 Determined automatically
1 Generate clique cuts moderately
2 Generate clique cuts aggressively
3 Generate clique cuts very aggressively

clocktype (integer): clock type for computation time
Decides how computation times are measured for both reporting performance and terminating optimization when
a time limit has been set. Small variations in measured time on identical runs may be expected on any computer
system with any setting of this parameter. The default setting 0 (zero) allows CPLEX to choose wall clock time
when other parameters invoke parallel optimization and to choose CPU time when other parameters enforce
sequential (not parallel) optimization. GAMS/Cplex works with wall time on Windows also for sequential
optimization unless this option instructs to measure time differently.
(default = 0)
0 Automatic
1 CPU time
2 Wall clock time

clonelog (integer): enable clone logs
The clone logs contain information normally recorded in the ordinary log file but inconvenient to send through
the normal log channel in case of parallel execution. The information likely to be of most interest to you are
special messages, such as error messages, that result from calls to the LP optimizers called for the subproblems.
The clone log files are named cloneK.log, where $K$ is the index of the clone, ranging from 0 (zero) to the number
of threads minus one. Since the clones are created at each call to a parallel optimizer and discarded when it exits,
the clone logs are opened at each call and closed at each exit. The clone log files are not removed when the
clones themselves are discarded.
(default = 0)
~1 Clone log files off
0 Automatic
1 Clone log files on

coeredind (integer): coefficient reduction on/off
Coefficient reduction is a technique used when presolving mixed integer programs. The benefit is to improve
the objective value of the initial (and subsequent) linear programming relaxations by reducing the number of
non-integral vertices. However, the linear programs generated at each node may become more difficult to solve.
(default = -1)

-1 Automatic
0 Do not use coefficient reduction
1 Reduce only to integral coefficients
2 Reduce all potential coefficients
3 Reduce aggressively with tilting

computeserver (string): address and port of Cplex remote object server

This option will use a remote machine to solve the model. The option is specified as name:port where name is the machine name or IP address of the remote server and port is the port number the Cplex remote server listen for work. On the remote server, a full Cplex installation (not just GAMS/Cplex) is required. On the server, one needs to start cplex -worker=tcpip -address=name:port with the same name/port as in this option.

covers (integer): cover cut generation

Determines whether or not cover cuts should be generated during optimization.

(default = 0)

-1 Do not generate cover cuts
0 Determined automatically
1 Generate cover cuts moderately
2 Generate cover cuts aggressively
3 Generate cover cuts very aggressively

craind (integer): crash strategy (used to obtain starting basis)

The crash option biases the way Cplex orders variables relative to the objective function when selecting an initial basis.

(default = 1)

-1 Primal: alternate ways of using objective coefficients. Dual: aggressive starting basis
0 Primal: ignore objective coefficients during crash. Dual: aggressive starting basis
1 Primal: alternate ways of using objective coefficients. Dual: default starting basis

cutlo (real): lower cutoff for tree search

Sets the lower cutoff tolerance. When the problem is a maximization problem, CPLEX cuts off or discards solutions that are less than the specified cutoff value. If the model has no solution with an objective value greater than or equal to the cutoff value, then CPLEX declares the model infeasible. In other words, setting the lower cutoff value c for a maximization problem is similar to adding this constraint to the objective function of the model: \( \text{obj} \geq c \).

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

(default = \(-1e+075\))

cutpass (integer): maximum number of cutting plane passes

Sets the upper limit on the number of passes that will be performed when generating cutting planes on a mixed integer model.

(default = 0)

-1 None
0 Automatically determined
>0 Maximum passes to perform
cuts (string): default cut generation

Allows generation setting of all optional cuts at once. This is done by changing the meaning of the default value (0: automatic) for the various Cplex cut generation options. The options affected are `cliques`, `covers`, `disjcuts`, `flowcovers`, `flowpaths`, `fraccuts`, `gubcovers`, `implbd`, `liftprojcuts`, `mcfcuts`, `mircuts`, and `symmetry`.

(default = 0)

-1 Do not generate cuts
0 Determined automatically
1 Generate cuts moderately
2 Generate cuts aggressively
3 Generate cuts very aggressively
4 Generate cuts highly aggressively
5 Generate cuts extremely aggressively

cutsfactor (real): cut limit

This option limits the number of cuts that can be added. The number of rows in the problem with cuts added is limited to `cutsfactor` times the original (after presolve) number of rows.

(default = 4)

cutup (real): upper cutoff for tree search

Sets the upper cutoff tolerance. When the problem is a minimization problem, CPLEX cuts off or discards any solutions that are greater than the specified upper cutoff value. If the model has no solution with an objective value less than or equal to the cutoff value, CPLEX declares the model infeasible. In other words, setting an upper cutoff value \( c \) for a minimization problem is similar to adding this constraint to the objective function of the model: \( \text{obj} \leq c \).

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

(default = 1e+075)

depind (integer): dependency checker on/off

This option determines if and when the dependency checker will be used.

(default = -1)

-1 Automatic
0 Turn off dependency checking
1 Turn on only at the beginning of preprocessing
2 Turn on only at the end of preprocessing
3 Turn on at the beginning and at the end of preprocessing

dettilim (real): deterministic time limit

Sets a time limit expressed in ticks, a unit to measure work done deterministically.

The length of a deterministic tick may vary by platform. Nevertheless, ticks are normally consistent measures for a given platform (combination of hardware and software) carrying the same load. In other words, the correspondence of ticks to clock time depends on the hardware, software, and the current load of the machine. For the same platform and same load, the ratio of ticks per second stays roughly constant, independent of the model solved. However, for very short optimization runs, the variation of this ratio is typically high.

(default = 1e+075)
**disjcuts** (*integer*): disjunctive cuts generation

Determines whether or not to generate disjunctive cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

(default = 0)

-1 Do not generate disjunctive cuts
0 Determined automatically
1 Generate disjunctive cuts moderately
2 Generate disjunctive cuts aggressively
3 Generate disjunctive cuts very aggressively

**divetype** (*integer*): MIP dive strategy

The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose. The default (automatic) setting chooses when to perform a probing dive, and the other two settings direct Cplex when to perform probing dives: never or always.

(default = 0)

0 Automatic
1 Traditional dive
2 Probing dive
3 Guided dive

**divflt** (*real*): solution pool range filter coefficients

A diversity filter for a solution pool (see option solnpool) allows you generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution.

A diversity filter drives the search for multiple solutions toward new solutions that satisfy a measure of diversity specified in the filter. This diversity measure applies only to binary variables. Potential new solutions are compared to a reference set. This reference set is specified with this dot option. If no reference set is specified, the difference measure will be computed relative to the other solutions in the pool. The diversity measure is computed by summing the pair-wise absolute differences from solution and the reference values.

(default = 0)

**divfltlo** (*real*): lower bound on diversity

Please check option .divflt for general information on a diversity filter.

If you specify a lower bound on the diversity using divfltlo, Cplex will look for solutions that are different from the reference values. In other words, you can say, Give me solutions that differ by at least this amount in this set of variables.

(default = mindouble)

**divfltup** (*real*): upper bound on diversity

Please check option .divflt for general information on a diversity filter.

If you specify an upper bound on diversity divfltup, Cplex will look for solutions similar to the reference values. In other words, you can say, Give me solutions that are close to this one, within this set of variables.

(default = maxdouble)

**dpriind** (*integer*): dual simplex pricing

Pricing strategy for dual simplex method. Consider using dual steepest-edge pricing. Dual steepest-edge is particularly efficient and does not carry as much computational burden as the primal steepest-edge pricing.

(default = 0)
0 Determined automatically
1 Standard dual pricing
2 Steepest-edge pricing
3 Steepest-edge pricing in slack space
4 Steepest-edge pricing, unit initial norms
5 Devex pricing

eachcutlim (integer): Sets a limit for each type of cut

This parameter allows you to set a uniform limit on the number of cuts of each type that Cplex generates. By default, the limit is a large integer; that is, there is no effective limit by default.

Tighter limits on the number of cuts of each type may benefit certain models. For example, a limit on each type of cut will prevent any one type of cut from being created in such large number that the limit on the total number of all types of cuts is reached before other types of cuts have an opportunity to be created. A setting of 0 means no cuts.

This parameter does not influence the number of Gomory cuts. For means to control the number of Gomory cuts, see also the fractional cut parameters: fraccand, fraccuts, and fracpass.

(default = 2100000000)

epagap (real): absolute stopping tolerance

Synonym: optca

Absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the epagap setting, the optimization is stopped. This option overrides GAMS OptCA which provides its initial value.

(default = GAMS OptCA)

epgap (real): relative stopping tolerance

Synonym: optcr

Relative tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the epgap setting, the mixed integer optimization is stopped. Note the difference in the Cplex definition of the relative tolerance with the GAMS definition. This option overrides GAMS OptCR which provides its initial value.

Range: [0, 1]

(default = GAMS OptCR)

epint (real): integrality tolerance

Integrality Tolerance. This specifies the amount by which an integer variable can be different than an integer and still be considered feasible.

Range: [0, 0.5]

(default = 1e-005)

epmrk (real): Markowitz pivot tolerance

The Markowitz tolerance influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.

Range: [0.0001, 0.99999]

(default = 0.01)

epopt (real): optimality tolerance

The optimality tolerance influences the reduced-cost tolerance for optimality. This option setting governs how closely Cplex must approach the theoretically optimal solution.
Range: $[1e^{-9}, 0.1]$
(default = $1e^{-6}$)

**epper (real):** perturbation constant

Perturbation setting. Highly degenerate problems tend to stall optimization progress. Cplex automatically perturbs the variable bounds when this occurs. Perturbation expands the bounds on every variable by a small amount thereby creating a different but closely related problem. Generally, the solution to the less constrained problem is easier to solve. Once the solution to the perturbed problem has advanced as far as it can go, Cplex removes the perturbation by resetting the bounds to their original values.

If the problem is perturbed more than once, the perturbation constant is probably too large. Reduce the *epper* option to a level where only one perturbation is required. Any value greater than or equal to 1.0e-8 is valid.

(default = $1e^{-6}$)

**eprhs (real):** feasibility tolerance

Feasibility tolerance. This specifies the degree to which a problem’s basic variables may violate their bounds. This tolerance influences the selection of an optimal basis and can be reset to a higher value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, Cplex may falsely conclude that a problem is infeasible.

Range: $[1e^{-9}, 0.1]$
(default = $1e^{-6}$)

**feasopt (integer):** computes a minimum-cost relaxation to make an infeasible model feasible

With *Feasopt* turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option *feasoptmode*.

Feasible relaxations are available for all problem types with the exception of quadratically constraint problems.

(default = 0)

0 Turns Feasible Relaxation off
1 Turns Feasible Relaxation on

**feasoptmode (integer):** Mode of FeasOpt

The parameter *FeasOptMode* allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter *FeasOptMode* indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations).

(default = 0)

0 Minimize sum of relaxations Minimize the sum of all required relaxations in first phase only
1 Minimize sum of relaxations and optimize Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations
2 Minimize number of relaxations Minimize the number of constraints and bounds requiring relaxation in first phase only
3 Minimize number of relaxations and optimize Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations
4 Minimize sum of squares of relaxations Minimize the sum of squares of required relaxations in first phase only

5 Minimize sum of squares of relaxations and optimize Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref (real): feasibility preference
You can express the costs associated with relaxing a bound or right hand side value during a feasopt run through the .feaspref option. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.
(default = 1)

flowcovers (integer): flow cover cut generation
Determines whether or not flow cover cuts should be generated during optimization.
(default = 0)
-1 Do not generate flow cover cuts
0 Determined automatically
1 Generate flow cover cuts moderately
2 Generate flow cover cuts aggressively

flowpaths (integer): flow path cut generation
Determines whether or not flow path cuts should be generated during optimization. At the default of 0, generation is continued only if it seems to be helping.
(default = 0)
-1 Do not generate flow path cuts
0 Determined automatically
1 Generate flow path cuts moderately
2 Generate flow path cuts aggressively

fpheur (integer): feasibility pump heuristic
Controls the use of the feasibility pump heuristic for mixed integer programming (MIP) models.
(default = 0)
-1 Turns Feasible Pump heuristic off
0 Automatic
1 Apply the feasibility pump heuristic with an emphasis on finding a feasible solution
2 Apply the feasibility pump heuristic with an emphasis on finding a feasible solution with a good objective value

fraccand (integer): candidate limit for generating Gomory fractional cuts
Limits the number of candidate variables for generating Gomory fractional cuts.
(default = 200)

fraccuts (integer): Gomory fractional cut generation
Determines whether or not Gomory fractional cuts should be generated during optimization.
(default = 0)
-1 Do not generate Gomory fractional cuts
0 Determined automatically
1 Generate Gomory fractional cuts moderately
2 Generate Gomory fractional cuts aggressively

**fracpass** (*integer*): maximum number of passes for generating Gomory fractional cuts

Sets the upper limit on the number of passes that will be performed when generating Gomory fractional cuts on a mixed integer model. Ignored if parameter *fraccuts* is set to a nonzero value.

(default = 0)

- 0 0 Automatically determined
- >0 Maximum passes to perform

**gubcovers** (*integer*): GUB cover cut generation

Determines whether or not GUB (Generalized Upper Bound) cover cuts should be generated during optimization.

The default of 0 indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.

(default = 0)

- -1 Do not generate GUB cover cuts
- 0 Determined automatically
- 1 Generate GUB cover cuts moderately
- 2 Generate GUB cover cuts aggressively

**heurfreq** (*integer*): heuristic frequency

This option specifies how often to apply the node heuristic. Setting to a positive number applies the heuristic at the requested node interval.

(default = 0)

- -1 Do not use the node heuristic
- 0 Determined automatically

**iis** (*integer*): run the IIS finder if the problem is infeasible

Find an IIS (Irreducably Inconsistent Set of constraints) and write an IIS report to the GAMS solution listing if the model is found to be infeasible. IIS is available for LP problems only.

(default = 0)

**implbd** (*integer*): implied bound cut generation

Determines whether or not implied bound cuts should be generated during optimization.

(default = 0)

- -1 Do not generate implied bound cuts
- 0 Determined automatically
- 1 Generate implied bound cuts moderately
- 2 Generate implied bound cuts aggressively

**interactive** (*integer*): allow interactive option setting after a Control-C

When set to yes, options can be set interactively after interrupting Cplex with a Control-C. Options are entered just as if they were being entered in the *cplex.opt* file. Control is returned to Cplex by entering continue. The optimization can be aborted by entering abort. This option can only be used when running from the command line. Moreover, the GAMS option InteractiveSolver needs to be set to 1.

(default = 0)

**intsollim** (*integer*): maximum number of integer solutions

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.

(default = large)


**itlim** (integer): iteration limit

Synonym: iterlim

The iteration limit option sets the maximum number of iterations before the algorithm terminates, without reaching optimality. This Cplex option overrides the GAMS IterLim option. Any non-negative integer value is valid.

(default = GAMS IterLim)

**lbheur** (integer): local branching heuristic

This parameter lets you control whether Cplex applies a local branching heuristic to try to improve new incumbents found during a MIP search. By default, this parameter is off. If you turn it on, Cplex will invoke a local branching heuristic only when it finds a new incumbent. If Cplex finds multiple incumbents at a single node, the local branching heuristic will be applied only to the last one found.

(default = 0)

- 0 Off
- 1 Apply local branching heuristic to new incumbent

**liftprojcuts** (integer): lift-and-project cuts

(default = 0)

- -1 Do not generate lift-and-project cuts
- 0 Determined automatically
- 1 Generate lift-and-project cuts moderately
- 2 Generate lift-and-project cuts aggressively
- 3 Generate lift-and-project cuts very aggressively

**localimplied** (integer): generation of locally valid implied bound cuts

(default = 0)

- -1 Do not generate locally valid implied bound cuts
- 0 Determined automatically
- 1 Generate locally valid implied bound cuts moderately
- 2 Generate locally valid implied bound cuts aggressively
- 3 Generate locally valid implied bound cuts very aggressively

**lpmethod** (integer): algorithm to be used for LP problems

Specifies which LP algorithm to use. If left at the default value (0 for automatic), and a primal-feasible basis is available, primal simplex will be used. If no primal-feasible basis is available, and threads is equal to 1, dual simplex will be used. If threads is greater than 1 and no primal-feasible basis is available, the concurrent option will be used.

Sifting may be useful for problems with many more variables than equations.

The concurrent option runs multiple methods in parallel. The first thread uses dual simplex. The second thread uses barrier. The next thread uses primal simplex. Remaining threads are used by the barrier run. If the aspect ratio (number of columns versus number of rows) is large, and if more than 10 threads are available, then concurrent optimization also invokes sifting on the LP. The solution is returned by first method to finish.

(default = 0)

- 0 Automatic
- 1 Primal Simplex
- 2 Dual Simplex
- 3 Network Simplex
mcfcuts (integer): multi-commodity flow cut generation

Specifies whether Cplex should generate multi-commodity flow (MCF) cuts in a problem where Cplex detects the characteristics of a multi-commodity flow network with arc capacities. By default, Cplex decides whether or not to generate such cuts. To turn off generation of such cuts, set this parameter to -1. Cplex is able to recognize the structure of a network as represented in many real-world models. When it recognizes such a network structure, Cplex is able to generate cutting planes that usually help solve such problems. In this case, the cuts that Cplex generates state that the capacities installed on arcs pointing into a component of the network must be at least as large as the total flow demand of the component that cannot be satisfied by flow sources within the component.

(default = 0)

-1 Do not generate MCF cuts
0 Determined automatically
1 Generate MCF cuts moderately
2 Generate MCF cuts aggressively

memoryemphasis (integer): Reduces use of memory

This parameter lets you indicate to Cplex that it should conserve memory where possible. When you set this parameter to its non default value, Cplex will choose tactics, such as data compression or disk storage, for some of the data computed by the barrier and MIP optimizers. Of course, conserving memory may impact performance in some models. Also, while solution information will be available after optimization, certain computations that require a basis that has been factored (for example, for the computation of the condition number Kappa) may be unavailable.

(default = 0)

0 Do not conserve memory
1 Conserve memory where possible

mipdisplay (integer): progress display level

The amount of information displayed during MIP solution increases with increasing values of this option.

(default = 4)

0 No display
1 Display integer feasible solutions
2 Displays nodes under mipinterval control
3 Same as 2 but adds information on cuts
4 Same as 3 but adds LP display for the root node
5 Same as 3 but adds LP display for all nodes

mipemphasis (integer): MIP solution tactics

This option controls the tactics for solving a mixed integer programming problem.

(default = 0)

0 Balance optimality and feasibility
1 Emphasize feasibility over optimality
2 Emphasize optimality over feasibility
3 Emphasize moving the best bound
4 Emphasize hidden feasible solutions

**mipinterval (integer):** progress display interval

Controls the frequency of node logging when the parameter **mipdisplay** is set higher than 1 (one). Frequency must be an integer; it may be 0 (zero), positive, or negative. By default, CPLEX displays new information in the node log during a MIP solve at relatively high frequency during the early stages of solving a MIP model, and adds lines to the log at progressively longer intervals as solving continues. In other words, CPLEX logs information frequently in the beginning and progressively less often as it works. When the value is a positive integer \( n \), CPLEX displays new incumbents, plus it displays a new line in the log every \( n \) nodes. When the value is a negative integer \( n \), CPLEX displays new incumbents, and the negative value determines how much processing CPLEX does before it displays a new line in the node log. A negative value close to zero means that CPLEX displays new lines in the log frequently. A negative value far from zero means that CPLEX displays new lines in the log less frequently. In other words, a negative value of this parameter contracts or dilates the interval at which CPLEX displays information in the node log.

(default = 0)

**mipkappastats (integer):** MIP kappa computation

MIP kappa summarizes the distribution of the condition number of the optimal bases CPLEX encountered during the solution of a MIP model. That summary may let you know more about the numerical difficulties of your MIP model. Because MIP kappa (as a statistical distribution) requires CPLEX to compute the condition number of the optimal bases of the subproblems during branch-and-cut search, you can compute the MIP kappa only when CPLEX solves the subproblem with its simplex optimizer. In other words, in order to obtain results with this parameter, you can not use the sifting optimizer nor the barrier without crossover to solve the subproblems. See the parameters **startalg** and **subalg**.

Computing the kappa of a subproblem has a cost. In fact, computing MIP kappa for the basis matrices can be computationally expensive and thus generally slows down the solution of a problem. Therefore, the setting 0 (automatic) tells CPLEX generally not to compute MIP kappa, but in cases where the parameter **numericalemphasis** is turned on, CPLEX computes MIP kappa for a sample of subproblems. The value 1 (sample) leads to a negligible performance degradation on average, but can slow down the branch-and-cut exploration by as much as 10% on certain models. The value 2 (full) leads to a 2% performance degradation on average, but can significantly slow the branch-and-cut exploration on certain models. In practice, the value 1 (sample) is a good trade-off between performance and accuracy of statistics. If you need very accurate statistics, then use value 2 (full).

In case CPLEX is instructed to compute a MIP kappa distribution, the parameter **quality** is automatically turned on.

(default = -1)

-1 No MIP kappa statistics; default
0 Automatic: let CPLEX decide
1 Compute MIP kappa for a sample of subproblems
2 Compute MIP kappa for all subproblems

**mipordind (integer):** priority list on/off

Synonym: prioropt

Use priorities. Priorities should be assigned based on your knowledge of the problem. Variables with higher priorities will be branched upon before variables of lower priorities. This direction of the tree search can often dramatically reduce the number of nodes searched. For example, consider a problem with a binary variable representing a yes/no decision to build a factory, and other binary variables representing equipment selections within that factory. You would naturally want to explore whether or not the factory should be built before considering what specific equipment to purchased within the factory. By assigning a higher priority to the build/no build decision variable, you can force this logic into the tree search and eliminate wasted computation time exploring uninteresting portions of the tree. When set at 0 (default), the **mipordind** option instructs Cplex not to use priorities for branching. When set to 1, priority orders are utilized.
Note: Priorities are assigned to discrete variables using the .prior suffix in the GAMS model. Lower .prior values mean higher priority. The .prioropt model suffix has to be used to signal GAMS to export the priorities to the solver.

(default = GAMS PriorOpt)

0 Do not use priorities for branching
1 Priority orders are utilized

mipordtype (integer): priority order generation
This option is used to select the type of generic priority order to generate when no priority order is present.

(default = 0)

0 None
1 decreasing cost magnitude
2 increasing bound range
3 increasing cost per coefficient count

mipsearch (integer): search strategy for mixed integer programs
Sets the search strategy for a mixed integer program. By default, Cplex chooses whether to apply dynamic search or conventional branch and cut based on characteristics of the model.

(default = 0)

0 Automatic
1 Apply traditional branch and cut strategy
2 Apply dynamic search

mipstart (integer): use mip starting values
This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the values should be checked to see if they provide an integer feasible solution before starting optimization.

(default = 0)

0 do not use the values
1 use the values

miqcpstrat (integer): MIQCP relaxation choice
This option controls how MIQCPs are solved. For some models, the setting 2 may be more effective than 1. You may need to experiment with this parameter to determine the best setting for your model.

(default = 0)

0 Automatic
1 QCP relaxation Cplex will solve a QCP relaxation of the model at each node.
2 LP relaxation Cplex will solve a LP relaxation of the model at each node.

mircuts (integer): mixed integer rounding cut generation
Determines whether or not to generate mixed integer rounding (MIR) cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

(default = 0)

-1 Do not generate MIR cuts
0 Determined automatically
1 Generate MIR cuts moderately
2 Generate MIR cuts aggressively
mpsLongNum (integer): MPS file format precision of numeric output

Determines the precision of numeric output in the MPS file formats. When this parameter is set to its default value 1 (one), numbers are written to MPS files in full-precision; that is, up to 15 significant digits may be written. The setting 0 (zero) writes files that correspond to the standard MPS format, where at most 12 characters can be used to represent a value. This limit may result in loss of precision.

(default = 1)

0 Use limited MPS precision
1 Use full-precision

names (integer): load GAMS names into Cplex

This option causes GAMS names for the variables and equations to be loaded into Cplex. These names will then be used for error messages, log entries, and so forth. Setting names to no may help if memory is very tight.

(default = 1)

netDisplay (integer): network display level

This option controls the log for network iterations.

(default = 2)

0 No network log.
1 Displays true objective values
2 Displays penalized objective values

netEpOpt (real): optimality tolerance for the network simplex method

This optimality tolerance influences the reduced-cost tolerance for optimality when using the network simplex method. This option setting governs how closely Cplex must approach the theoretically optimal solution.

Range: [1e-011, 0.1]

(default = 1e-006)

netEprhs (real): feasibility tolerance for the network simplex method

This feasibility tolerance determines the degree to which the network simplex algorithm will allow a flow value to violate its bounds.

Range: [1e-011, 0.1]

(default = 1e-006)

netFind (integer): attempt network extraction

Specifies the level of network extraction to be done.

(default = 2)

1 Extract pure network only
2 Try reflection scaling
3 Try general scaling

netItLim (integer): iteration limit for network simplex

Iteration limit for the network simplex method.

(default = large)

netPpriInd (integer): network simplex pricing

Network simplex pricing algorithm. The default of 0 (currently equivalent to 3) shows best performance for most problems.

(default = 0)

0 Automatic
1 Partial pricing
2 Multiple partial pricing
3 Multiple partial pricing with sorting

**nodefileind (integer):** node storage file indicator

Specifies how node files are handled during MIP processing. Used when parameter `workmem` has been exceeded by the size of the branch and cut tree. If set to 0 when the tree memory limit is reached, optimization is terminated. Otherwise a group of nodes is removed from the in-memory set as needed. By default, Cplex transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node files in compressed form in memory. At settings 2 and 3, the node files are transferred to disk. They are stored under a directory specified by parameter `workdir` and Cplex actively manages which nodes remain in memory for processing. (default = 1)

0 No node files
1 Node files in memory and compressed
2 Node files on disk
3 Node files on disk and compressed

**nodelim (integer):** maximum number of nodes to solve

Synonym: nodlim

The maximum number of nodes solved before the algorithm terminates, without reaching optimality. This option overrides the GAMS NodLim model suffix. When this parameter is set to 0 (this is only possible through an option file), Cplex completes processing at the root; that is, it creates cuts and applies heuristics at the root. When this parameter is set to 1 (one), it allows branching from the root; that is, nodes are created but not solved. (default = GAMS NodLim)

**nodesel (integer):** node selection strategy

This option is used to set the rule for selecting the next node to process when backtracking. (default = 1)

0 Depth-first search This chooses the most recently created node.
1 Best-bound search This chooses the unprocessed node with the best objective function for the associated LP relaxation.
2 Best-estimate search This chooses the node with the best estimate of the integer objective value that would be obtained once all integer infeasibilities are removed.
3 Alternate best-estimate search

**numericalemphasis (integer):** emphasizes precision in numerically unstable or difficult problems

This parameter lets you indicate to Cplex that it should emphasize precision in numerically difficult or unstable problems, with consequent performance trade-offs in time and memory. (default = 0)

0 Off
1 Exercise extreme caution in computation

**objdif (real):** overrides GAMS Cheat parameter

Synonym: cheat

A means for automatically updating the cutoff to more restrictive values. Normally the most recently found integer feasible solution objective value is used as the cutoff for subsequent nodes. When this option is set to a positive value, the value will be subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the MIP optimization to ignore integer solutions that are not at least this amount better than the one found so far. The option can be adjusted to improve problem solving efficiency by
limiting the number of nodes; however, setting this option at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. Negative values for this option will result in some integer solutions that are worse than or the same as those previously generated, but will not necessarily result in the generation of all possible integer solutions. This option overrides the GAMS Cheat parameter.

\(\text{(default = 0)}\)

**objlim (real):** objective function lower limit

Setting a lower objective function limit will cause Cplex to halt the optimization process once the minimum objective function value limit has been exceeded.

\(\text{(default = -1e+075)}\)

**objrng (string):** do objective ranging

Calculate sensitivity ranges for the specified GAMS variables. Unlike most options, objrng can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS variable named. Specifying all will cause range information to be produced for all variables. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option rngrestart is specified.

\(\text{(default = no objective ranging is done)}\)

**objulim (real):** objective function upper limit

Setting an upper objective function limit will cause Cplex to halt the optimization process once the maximum objective function value limit has been exceeded.

\(\text{(default = 1e+075)}\)

**parallelmode (integer):** parallel optimization mode

Sets the parallel optimization mode. Possible modes are automatic, deterministic, and opportunistic.

In this context, deterministic means that multiple runs with the same model at the same parameter settings on the same platform will reproduce the same solution path and results. In contrast, opportunistic implies that even slight differences in timing among threads or in the order in which tasks are executed in different threads may produce a different solution path and consequently different timings or different solution vectors during optimization executed in parallel threads. When running with multiple threads, the opportunistic setting entails less synchronization between threads and consequently may provide better performance.

In deterministic mode, Cplex applies as much parallelism as possible while still achieving deterministic results. That is, when you run the same model twice on the same platform with the same parameter settings, you will see the same solution and optimization run.

More opportunities to exploit parallelism are available if you do not require determinism. In other words, Cplex can find more opportunities for parallelism if you do not require an invariant, repeatable solution path and precisely the same solution vector. To use all available parallelism, you need to select the opportunistic parallel mode. In this mode, Cplex will utilize all opportunities for parallelism in order to achieve best performance.

However, in opportunistic mode, the actual optimization may differ from run to run, including the solution time itself. A truly parallel deterministic algorithm is available only for MIP optimization. Only opportunistic parallel algorithms (barrier and concurrent optimizers) are available for continuous models. (Each of the simplex algorithms runs sequentially on a continuous model.) Consequently, when parallel mode is set to deterministic, both barrier and concurrent optimizers are restricted to run only sequentially, not in parallel.

A GAMS/Cplex run will use deterministic mode unless explicitly specified.

If parallelmode is explicitly set to 0 (automatic) the settings of this parallel mode parameter interact with settings of the threads parameter. Let the result number of threads available to Cplex be \(n\) (note that negative values for the threads parameter are possible to exclude work on some cores).

\(n=0:\) Cplex uses maximum number of threads (determined by the computing platform) in deterministic mode unless parallelmode is set to -1 (opportunistic).

\(n=1:\) Cplex runs sequential.

\(n>1:\) Cplex uses maximum number of threads (determined by the computing platform) in opportunistic mode
unless `parallelmode` is set to 1 (deterministic).

Here is a list of possible values:

(default = 1)

-1 Enable opportunistic parallel search mode
0 Automatic
1 Enable deterministic parallel search mode

**perind (integer):** force initial perturbation

Perturbation Indicator. If a problem automatically perturbs early in the solution process, consider starting the solution process with a perturbation by setting `perind` to 1. Manually perturbing the problem will save the time of first allowing the optimization to stall before activating the perturbation mechanism, but is useful only rarely, for extremely degenerate problems.

(default = 0)

0 not automatically perturbed
1 automatically perturbed

**perlim (integer):** number of stalled iterations before perturbation

Perturbation limit. The number of stalled iterations before perturbation is invoked. The default value of 0 means the number is determined automatically.

(default = 0)

**polishafterdettime (real):** Deterministic time before starting to polish a feasible solution

(default = 1e+075)

**polishafterepgap (real):** Absolute MIP gap before starting to polish a feasible solution

Sets an absolute MIP gap (that is, the difference between the best integer objective and the objective of the best node remaining) after which CPLEX stops branch-and-cut and begins polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

(default = 0)

**polishafterpgap (real):** Relative MIP gap before starting to polish a feasible solution

Sets a relative MIP gap after which CPLEX will stop branch-and-cut and begin polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

(default = 0)

**polishafterintsol (integer):** MIP integer solutions to find before starting to polish a feasible solution

Sets the number of integer solutions to find before CPLEX stops branch-and-cut and begins to polish a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

(default = 2100000000)

**polishafternode (integer):** Nodes to process before starting to polish a feasible solution

Sets the number of nodes processed in branch-and-cut before CPLEX starts solution polishing, if a feasible
solution is available.
(default = 2100000000)

polishaftertime (real): Time before starting to polish a feasible solution

Tells CPLEX how much time in seconds to spend during mixed integer optimization before CPLEX starts polishing a feasible solution. The default value is such that CPLEX does not start solution polishing by default.

(default = 1e+075)

populatelim (integer): limit of solutions generated for the solution pool by populate method

Limits the number of solutions generated for the solution pool during each call to the populate procedure. Populate stops when it has generated PopulateLim solutions. A solution is counted if it is valid for all filters (see .divflt and consistent with the relative and absolute pool gap parameters (see solnpoolgap and solnpoolagap), and has not been rejected by the incumbent checking routine (see userincbcall), whether or not it improves the objective of the model. This parameter does not apply to MIP optimization generally; it applies only to the populate procedure.

If you are looking for a parameter to control the number of solutions stored in the solution pool, consider the parameter solnpoolemit instead.

Populate will stop before it reaches the limit set by this parameter if it reaches another limit, such as a time or node limit set by the user.

(default = 20)

ppriind (integer): primal simplex pricing

Pricing algorithm. Likely to show the biggest impact on performance. Look at overall solution time and the number of Phase I and total iterations as a guide in selecting alternate pricing algorithms. If you are using the dual Simplex method use dpriind to select a pricing algorithm. If the number of iterations required to solve your problem is approximately the same as the number of rows in your problem, then you are doing well. Iteration counts more than three times greater than the number of rows suggest that improvements might be possible.

(default = 0)

- 1 Reduced-cost pricing This is less compute intensive and may be preferred if the problem is small or easy. This option may also be advantageous for dense problems (say 20 to 30 nonzeros per column).

0 Hybrid reduced-cost and Devex pricing

1 Devex pricing This may be useful for more difficult problems which take many iterations to complete Phase I. Each iteration may consume more time, but the reduced number of total iterations may lead to an overall reduction in time. Tenfold iteration count reductions leading to threefold speed improvements have been observed. Do not use devex pricing if the problem has many columns and relatively few rows. The number of calculations required per iteration will usually be disadvantageous.

2 Steepest edge pricing If devex pricing helps, this option may be beneficial. Steepest-edge pricing is computationally expensive, but may produce the best results on exceptionally difficult problems.

3 Steepest edge pricing with slack initial norms This reduces the computationally intensive nature of steepest edge pricing.

4 Full pricing

predual (integer): give dual problem to the optimizer

Solve the dual. Some linear programs with many more rows than columns may be solved faster by explicitly solving the dual. The predual option will cause Cplex to solve the dual while returning the solution in the context of the original problem. This option is ignored if presolve is turned off.

(default = 0)

- 1 do not give dual to optimizer

0 automatic

1 give dual to optimizer
**preind (integer)**: turn presolver on/off

Perform Presolve. This helps most problems by simplifying, reducing and eliminating redundancies. However, if there are no redundancies or opportunities for simplification in the model, it may be faster to turn presolve off to avoid this step. On rare occasions, the presolved model, although smaller, may be more difficult than the original problem. In this case turning the presolve off leads to better performance. Specifying 0 turns the aggregator off as well.

(default = 1)

**prelinear (integer)**: linear reduction indicator

If only linear reductions are performed, each variable in the original model can be expressed as a linear form of variables in the presolved model.

(default = 1)

**prepass (integer)**: number of presolve applications to perform

Number of MIP presolve applications to perform. By default, Cplex determines this automatically. Specifying 0 turns off the presolve but not the aggregator. Set **preind** to 0 to turn both off.

(default = -1)

-1 Determined automatically
0 No presolve

**preslvnd (integer)**: node presolve selector

Indicates whether node presolve should be performed at the nodes of a mixed integer programming solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective.

(default = 0)

-1 No node presolve
0 Automatic
1 Force node presolve
2 Perform probing on integer-infeasible variables

**pricelim (integer)**: pricing candidate list

Size for the pricing candidate list. Cplex dynamically determines a good value based on problem dimensions. Only very rarely will setting this option manually improve performance. Any non-negative integer values are valid.

(default = 0, in which case it is determined automatically)

**printfdoors (integer)**: list values of all options to GAMS listing file

Write the values of all options to the GAMS listing file. Valid values are no or yes.

(default = 0)

**probe (integer)**: perform probing before solving a MIP

Determines the amount of probing performed on a MIP. Probing can be both very powerful and very time consuming. Setting the value to 1 can result in dramatic reductions or dramatic increases in solution time depending on the particular model.

(default = 0)

-1 No probing
0 Automatic
1 Limited probing
2 More probing
3 Full probing
**probedettime** (*real*): deterministic time spent probing

(default = $1e+075$)

**probetime** (*real*): time spent probing

Limits the amount of time in seconds spent probing.

(default = $1e+075$)

**qpmakepsdind** (*integer*): adjust MIQP formulation to make the quadratic matrix positive-semi-definite

Determines whether Cplex will attempt to adjust a MIQP formulation, in which all the variables appearing in the quadratic term are binary. When this feature is active, adjustments will be made to the elements of a quadratic matrix that is not nominally positive semi-definite (PSD, as required by Cplex for all QP formulations), to make it PSD, and will also attempt to tighten an already PSD matrix for better numerical behavior. The default setting of 1 means yes but you can turn it off if necessary; most models should benefit from the default setting.

(default = 1)

0 Off

1 On

**qpmethod** (*integer*): algorithm to be used for QP problems

Specifies which QP algorithm to use.

At the default of 0 (automatic), barrier is used for QP problems and dual simplex for the root relaxation of MIQP problems.

=default = 0\)

0 Automatic

1 Primal Simplex

2 Dual Simplex

3 Network Simplex

4 Barrier

5 Sifting

6 Concurrent dual, barrier, and primal

**qtolin** (*integer*): linearization of the quadratic terms in the objective function of a QP or MIQP model

This parameter switches on or off linearization of the quadratic terms in the objective function of a quadratic program or of a mixed integer quadratic program.

In a convex mixed integer quadratic program, this parameter controls whether Cplex linearizes the product of binary variables in the objective function during presolve. In a nonconvex quadratic program or mixed integer quadratic program solved to global optimality according to solutiontarget, this parameter controls how Cplex linearizes the product of bounded variables in the objective function during presolve.

This parameter interacts with the existing parameter solutiontarget: When the solution target type is set to 1 (that is, Cplex searches for a globally optimal solution to a convex model), then in a convex MIQP, this parameter tells Cplex to replace the product of a binary variable and a bounded linear variable by a linearly constrained variable. When the solution target type is set to 3, then in a nonconvex QP or nonconvex MIQP, this parameter controls the initial relaxation.

(default = -1)

-1 Automatic

0 Off, Cplex does not linearize quadratic terms in the objective

1 On, Cplex linearizes quadratic terms in the objective

**quality** (*integer*): write solution quality statistics
Write solution quality statistics to the listing file. If set to yes, the statistics appear after the Solve Summary and before the Solution Listing.
(default = 0)

randomseed (integer): sets the random seed differently for diversity of solutions
(default = changes with each Cplex release)

readflt (string): reads Cplex solution pool filter file
The GAMS/Cplex solution pool options cover the basic use of diversity and range filters for producing multiple solutions. If you need multiple filters, weights on diversity filters or other advanced uses of solution pool filters, you could produce a Cplex filter file with your favorite editor or the GAMS Put Facility and read this into GAMS/Cplex using this option.

reduce (integer): primal and dual reduction type
Determines whether primal reductions, dual reductions, or both, are performed during preprocessing. It is occasionally advisable to do only one or the other when diagnosing infeasible or unbounded models.
(default = 3)
0 No primal or dual reductions
1 Only primal reductions
2 Only dual reductions
3 Both primal and dual reductions

reinv (integer): refactorization frequency
Refactorization Frequency. This option determines the number of iterations between refactorizations of the basis matrix. The default should be optimal for most problems. Cplex’s performance is relatively insensitive to changes in refactorization frequency. Only for extremely large, difficult problems should reducing the number of iterations between refactorizations be considered. Any non-negative integer value is valid.
(default = 0, in which case it is determined automatically)

relaxfixedinfeas (integer): access small infeasibilties in the solve of the fixed problem
Sometimes the solution of the fixed problem of a MIP does not solve to optimality due to small (dual) infeasibilities. The default behavior of the GAMS/Cplex link is to return the primal solution values only. If the option is set to 1, the small infeasibilities are ignored and a full solution including the dual values are reported back to GAMS.
(default = 0)
0 Off
1 On

relaxpreind (integer): presolve for initial relaxation on/off
This option will cause the Cplex presolve to be invoked for the initial relaxation of a mixed integer program (according to the other presolve option settings). Sometimes, additional reductions can be made beyond any MIP presolve reductions that may already have been done.
(default = -1)
-1 Automatic
0 do not presolve initial relaxation
1 use presolve on initial relaxation

relobjdif (real): relative cheat parameter
The relative version of the objdif option. Ignored if objdif is non-zero.
(default = 0)
repairtries (integer): try to repair infeasible MIP start

This parameter lets you indicate to Cplex whether and how many times it should try to repair an infeasible MIP start that you supplied. The parameter has no effect if the MIP start you supplied is feasible. It has no effect if no MIP start was supplied.

(default = 0)

-1 None: do not try to repair
0 Automatic
>0 Maximum tries to perform

repeatpresolve (integer): reapply presolve at root after preprocessing

This integer parameter tells Cplex whether to re-apply presolve, with or without cuts, to a MIP model after processing at the root is otherwise complete.

(default = -1)

-1 Automatic
0 Turn off represolve
1 Represolve without cuts
2 Represolve with cuts
3 Represolve with cuts and allow new root cuts

rerun (string): rerun problem if presolve infeasible or unbounded

The Cplex presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Cplex can, in order to get better diagnostic information, rerun the problem with presolve turned off. The GAMS solution listing will then mark variables and equations as infeasible or unbounded according to the final solution returned by the simplex algorithm. The iis option can be used to get even more diagnostic information. The rerun option controls this behavior. Valid values are auto, yes, no and nono. The value of auto is equivalent to no if names are successfully loaded into Cplex and option iis is set to no. In that case the Cplex messages from presolve help identify the cause of infeasibility or unboundedness in terms of GAMS variable and equation names. If names are not successfully loaded, rerun defaults to yes. Loading of GAMS names into Cplex is controlled by option names. The value of nono only affects MIP models for which Cplex finds a feasible solution in the branch-and-bound tree but the fixed problem turns out to be infeasible. In this case the value nono also disables the rerun without presolve, while the value of no still tries this run. Feasible integer solution but an infeasible fixed problem happens in few cases and mostly with badly scaled models. If you experience this try more aggressive scaling (scaind) or tightening the integer feasibility tolerance epint. If the fixed model is infeasible only the primal solution is returned to GAMS. You can recognize this inside GAMS by checking the marginal of the objective defining constraint which is always nonzero.

(default = yes)

auto Automatic
yes Rerun infeasible models with presolve turned off
no Do not rerun infeasible models
nono Do not rerun infeasible fixed MIP models

rhsrng (string): do right-hand-side ranging

Calculate sensitivity ranges for the specified GAMS equations. Unlike most options, rhsrng can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS equation named. Specifying all will cause range information to be produced for all equations. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option rngrestart is specified.

(default = no right-hand-side ranging is done)

rinsheur (integer): relaxation induced neighborhood search frequency

Cplex implements a heuristic known a Relaxation Induced Neighborhood Search (RINS) for MIP and MIQCP
problems. RINS explores a neighborhood of the current incumbent to try to find a new, improved incumbent. It formulates the neighborhood exploration as a MIP, a subproblem known as the subMIP, and truncates the subMIP solution by limiting the number of nodes explored in the search tree.

Parameter *rinsheur* controls how often RINS is invoked. A value of 100, for example, means that RINS is invoked every hundredth node in the tree.

(default = 0)

-1 Disable RINS
0 Automatic

**rngrestart (string):** write GAMS readable ranging information file

Write ranging information, in GAMS readable format, to the file named. Options *objrng* and *rhsrng* are used to specify which GAMS variables or equations are included.

(default = ranging information is printed to the listing file)

**scaind (integer):** matrix scaling on/off

This option influences the scaling of the problem matrix.

(default = 0)

-1 No scaling
0 Standard scaling An equilibration scaling method is implemented which is generally very effective.
1 Modified, more aggressive scaling method This method can produce improvements on some problems. This scaling should be used if the problem is observed to have difficulty staying feasible during the solution process.

**siftalg (integer):** sifting subproblem algorithm

Sets the algorithm to be used for solving sifting subproblems.

(default = 0)

0 Automatic
1 Primal simplex
2 Dual simplex
3 Network simplex
4 Barrier

**siftdisplay (integer):** sifting display level

Determines the amount of sifting progress information to be displayed.

(default = 1)

0 No display
1 Display major iterations
2 Display LP subproblem information

**siftitlim (integer):** limit on sifting iterations

Sets the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.

(default = large)

**simdisplay (integer):** simplex display level

This option controls what Cplex reports (normally to the screen) during optimization. The amount of information displayed increases as the setting value increases.

(default = 1)
0 No iteration messages are issued until the optimal solution is reported
1 An iteration log message will be issued after each refactorization Each entry will contain the
iteration count and scaled infeasibility or objective value.
2 An iteration log message will be issued after each iteration The variables, slacks and artificials
entering and leaving the basis will also be reported.

**singlim (integer):** limit on singularity repairs

The singularity limit setting restricts the number of times Cplex will attempt to repair the basis when singularities
are encountered. Once the limit is exceeded, Cplex replaces the current basis with the best factorizable basis that
has been found. Any non-negative integer value is valid.

(default = 10)

**solnpool (string):** solution pool file name

The solution pool enables you to generate and store multiple solutions to a MIP problem. The option expects a
GDX filename. This GDX file name contains the information about the different solutions generated by Cplex.
Inside your GAMS program you can process the GDX file and read the different solution point files. Please
check the GAMS/Cplex solver guide document and the example model `solnpool.gms` from the GAMS model
library.

**solnpoolagap (real):** absolute tolerance for the solutions in the solution pool

Sets an absolute tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse
(either greater in the case of a minimization, or less in the case of a maximization) than the objective of the
incumbent solution according to this measure are not kept in the solution pool.

Values of the solution pool absolute gap and the solution pool relative gap `solnpoolgap` may differ: For
example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and
also within 1\% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool
only if it is valid for both the relative and the absolute gaps.

The solution pool absolute gap parameter can also be used as a stopping criterion for the populate procedure: if
populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence
of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is
reached.

(default = 1e+075)

**solnpoolcapacity (integer):** limits of solutions kept in the solution pool

Limits the number of solutions kept in the solution pool. At most, `solnpoolcapacity` solutions will be stored
in the pool. Superfluous solutions are managed according to the replacement strategy set by the solution pool
replacement parameter `solnpoolreplace`.

The optimization (whether by MIP optimization or the populate procedure) will not stop if more than `sol-
npoolcapacity` are generated. Instead, stopping criteria are regular node and time limits and `populatelimit`,
`solnpoolgap` and `solnpoolagap`.

(default = 2100000000)

**solnpoolgap (real):** relative tolerance for the solutions in the solution pool

Sets a relative tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse
(either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution
by this measure are not kept in the solution pool.

Values of the solution pool absolute gap `solnpoolagap` and the solution pool relative gap may differ: For
example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and
within 1\% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if
it is valid for both the relative and the absolute gaps.

The solution pool relative gap parameter can also be used as a stopping criterion for the populate procedure: if
populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence
of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is
reached.
(default = 1e+075)

**solnpoolintensity (integer):** solution pool intensity for ability to produce multiple solutions

Controls the trade-off between the number of solutions generated for the solution pool and the amount of time or memory consumed. This parameter applies both to MIP optimization and to the populate procedure.

Values from 1 to 4 invoke increasing effort to find larger numbers of solutions. Higher values are more expensive in terms of time and memory but are likely to yield more solutions.

(default = 0)

0 Automatic Its default value, 0, lets Cplex choose which intensity to apply.

1 Mild: generate few solutions quickly For value 1, the performance of MIP optimization is not affected. There is no slowdown and no additional consumption of memory due to this setting. However, populate will quickly generate only a small number of solutions. Generating more than a few solutions with this setting will be slow. When you are looking for a larger number of solutions, use a higher value of this parameter.

2 Moderate: generate a larger number of solutions For value 2, some information is stored in the branch and cut tree so that it is easier to generate a larger number of solutions. This storage has an impact on memory used but does not lead to a slowdown in the performance of MIP optimization. With this value, calling populate is likely to yield a number of solutions large enough for most purposes. This value is a good choice for most models.

3 Aggressive: generate many solutions and expect performance penalty For value 3, the algorithm is more aggressive in computing and storing information in order to generate a large number of solutions. Compared to values 1 and 2, this value will generate a larger number of solutions, but it will slow MIP optimization and increase memory consumption. Use this value only if setting this parameter to 2 does not generate enough solutions.

4 Very aggressive: enumerate all practical solutions For value 4, the algorithm generates all solutions to your model. Even for small models, the number of possible solutions is likely to be huge; thus enumerating all of them will take time and consume a large quantity of memory.

**solnpoolmerge (string):** solution pool file name for merged solutions

Similar to `solnpooldump` this option enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file contains all variables with an additional first index (determined through `solnpoolprefix`) as parameters (Cplex only reports the primal solution). Inside your GAMS program you can process the GDX file and read all solutions in one read operation. Please check the GAMS/Cplex solver guide document for further solution pool options and the example model `solnpool.gms` from the GAMS model library.

**solnpoolnumsym (integer):** maximum number of variable symbols when writing merged solutions

(default = 10)

**solnpoolpop (integer):** methods to populate the solution pool

Regular MIP optimization automatically adds incumbents to the solution pool as they are discovered. Cplex also provides a procedure known as `populate` specifically to generate multiple solutions. You can invoke this procedure either as an alternative to the usual MIP optimizer or as a successor to the MIP optimizer. You can also invoke this procedure many times in a row in order to explore the solution space differently (see option `solnpoolpoprepeat`). In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory.

(default = 1)

1 Just collect the incumbents found during regular optimization

2 Calls the populate procedure

**solnpoolpopdel (string):** file with solution numbers to delete from the solution pool
After the GAMS program specified in `solnpoolpoprepeat` determined to continue the search for alternative solutions, the file specified by this option is read in. The solution numbers present in this file will be deleted from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

`solnpoolpoprepeat (string)`: method to decide if populating the solution should be repeated

After the termination of the populate procedure (see option `solnpoolpop`). The GAMS program specified in this option will be called which can examine the solutions in the solution pool and can decide to run the populate procedure again. If the GAMS program terminates normally (not compilation or execution time error) the search for new alternative solutions will be repeated.

`solnpoolprefix (string)`: file name prefix for GDX solution files

(default = `soln`)

`solnpoolreplace (integer)`: strategy for replacing a solution in the solution pool

(default = 0)

0 Replace the first solution (oldest) by the most recent solution; first in, first out
1 Replace the solution which has the worst objective
2 Replace solutions in order to build a set of diverse solutions

`solutiontarget (integer)`: type of solution when solving a nonconvex continuous quadratic model

This parameter specifies the type of solution when solving a nonconvex, continuous quadratic model. This parameter affects the behavior only when CPLEX uses the barrier algorithm without crossover to solve a nonconvex continuous quadratic model (QP); that is, the variables of the model are continuous, the objective function includes a quadratic term, and the objective function is not positive semi-definite (PSD).

(default = 0)

0 Automatic CPLEX first attempts to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX will return with an error (Q is not PSD).
1 Search for a globally optimal solution to a convex model
2 Search for a solution that satisfies first-order optimality conditions no optimality guarantee CPLEX first attempt to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX searches for a solution that satisfies first-order optimality conditions but is not necessarily globally optimal.
3 Search for a globally optimal solution regardless of convexity

`solvefinal (integer)`: switch to solve the problem with fixed discrete variables

Sometimes the solution process after the branch-and-cut that solves the problem with fixed discrete variables takes a long time and the user is interested in the primal values of the solution only. In these cases, `solvefinal` can be used to turn this final solve off. Without the final solve no proper marginal values are available and only zeros are returned to GAMS.

(default = 1)

0 Do not solve the fixed problem
1 Solve the fixed problem and return duals

`startalg (integer)`: MIP starting algorithm

Selects the algorithm to use for the initial relaxation of a MIP.

(default = 0)

0 Automatic
1 Primal simplex
2 Dual simplex
3 Network simplex
4 Barrier
5 Sifting
6 Concurrent

**strongcandlim (integer):** size of the candidates list for strong branching

Limit on the length of the candidate list for strong branching ($\text{varsel} = 3$).

(default = 10)

**strongitlim (integer):** limit on iterations per branch for strong branching

Limit on the number of iterations per branch in strong branching ($\text{varsel} = 3$). The default value of 0 causes the limit to be chosen automatically which is normally satisfactory. Try reducing this value if the time per node seems excessive. Try increasing this value if the time per node is reasonable but Cplex is making little progress.

(default = 0)

**subalg (integer):** algorithm for subproblems

Strategy for solving linear sub-problems at each node.

(default = 0)

0 Automatic
1 Primal simplex
2 Dual simplex
3 Network optimizer followed by dual simplex
4 Barrier with crossover
5 Sifting

**submipnodelim (integer):** limit on number of nodes in an RINS subMIP

Controls the number of nodes explored in an RINS subMIP. See option $\text{rinsheur}$.

(default = 500)

**symmetry (integer):** symmetry breaking cuts

Determines whether symmetry breaking cuts may be added, during the preprocessing phase, to a MIP model.

(default = -1)

-1 Automatic
0 Turn off symmetry breaking
1 Moderate level of symmetry breaking
2 Aggressive level of symmetry breaking
3 Very aggressive level of symmetry breaking
4 Highly aggressive level of symmetry breaking
5 Extremely aggressive level of symmetry breaking

**threads (integer):** global default thread count

Synonym: gthreads

Default number of parallel threads allowed for any solution method. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks. Cplex does not understand negative values for the threads parameter. GAMS/Cplex will translate this is a non-negative number by applying the following formula: $\text{max}(1, \text{number of cores}-|\text{threads}|)$

(default = GAMS Threads)
tilim (real): overrides the GAMS ResLim option

Synonym: reslim

The time limit setting determines the amount of time in seconds that Cplex will continue to solve a problem. This Cplex option overrides the GAMS ResLim option. Any non-negative value is valid.
(default = GAMS ResLim)

trelim (real): maximum space in memory for tree

Sets an absolute upper limit on the size (in megabytes) of the branch and cut tree. If this limit is exceeded, Cplex terminates optimization.
(default = 1e+075)

tuning (string): invokes parameter tuning tool

Invokes the Cplex parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Cplex option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in LP, MPS or SAV format. Cplex will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Cplex option file name without considering the problem provided by GAMS (use option writesav to create a SAV file of the problem provided by GAMS and include this name in the list of problems). The result of such a run is the updated GAMS/Cplex option file with a tuned set of parameters. The solver and model status returned to GAMS will be NORMAL COMPLETION and NO SOLUTION. More details on Cplex tuning can be found on IBM’s web page. Tuning is incompatible with the BCH facility and other advanced features of GAMS/Cplex.

tuningdettilim (real): tuning deterministic time limit per model or suite

(default = 1e+007)

tuningdisplay (integer): level of information reported by the tuning tool

Specifies the level of information reported by the tuning tool as it works.
(default = 1)

  0 Turn off display
  1 Display standard minimal reporting
  2 Display standard report plus parameter settings being tried
  3 Display exhaustive report and log

tuningmeasure (integer): measure for evaluating progress for a suite of models

Controls the measure for evaluating progress when a suite of models is being tuned. Choices are mean average and minmax of time to compare different parameter sets over a suite of models.
(default = 1)

  1 mean average
  2 minmax

tuningrepeat (integer): number of times tuning is to be repeated on perturbed versions

Specifies the number of times tuning is to be repeated on perturbed versions of a given problem. The problem is perturbed automatically by Cplex permuting its rows and columns. This repetition is helpful when only one problem is being tuned, as repeated perturbation and re-tuning may lead to more robust tuning results. This parameter applies to only one problem in a tuning session.
(default = 1)

tuningtilim (real): tuning time limit per model or suite

Sets a time limit per model and per test set (that is, suite of models).

As an example, suppose that you want to spend an overall amount of time tuning the parameter settings for a
given model, say, 2000 seconds. Also suppose that you want Cplex to make multiple attempts within that overall time limit to tune the parameter settings for your model. Suppose further that you want to set a time limit on each of those attempts, say, 200 seconds per attempt. In this case you need to specify an overall time limit of 2000 using GAMS option `reslim` or Cplex option `tilim` and `tuningtilim` to 200.

\[(default = 0.2 \times \text{GAMS ResLim})\]

**userincbcall (string):** The GAMS command line to call the incumbent checking program

The GAMS command line (minus the GAMS executable name) to call the incumbent checking routine. The incumbent is rejected if the GAMS program terminates normally. In case of a compilation or execution error, the incumbent is accepted.

**varsel (integer):** variable selection strategy at each node

This option is used to set the rule for selecting the branching variable at the node which has been selected for branching. The default value of 0 allows Cplex to select the best rule based on the problem and its progress.

\[(default = 0)\]

-1 Branch on variable with minimum infeasibility This rule may lead more quickly to a first integer feasible solution, but will usually be slower overall to reach the optimal integer solution.

0 Branch variable automatically selected

1 Branch on variable with maximum infeasibility This rule forces larger changes earlier in the tree, which tends to produce faster overall times to reach the optimal integer solution.

2 Branch based on pseudo costs Generally, the pseudo-cost setting is more effective when the problem contains complex trade-offs and the dual values have an economic interpretation.

3 Strong Branching This setting causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is most promising. This is often effective on large, difficult problems.

4 Branch based on pseudo reduced costs

**workdir (string):** directory for working files

The name of an existing directory into which Cplex may store temporary working files. Used for MIP node files and by out-of-core Barrier.

\[(default = \text{current or project directory})\]

**workmem (real):** memory available for working storage

Upper limit on the amount of memory, in megabytes, that Cplex is permitted to use for working files. See parameter `workdir`.

\[(default = 128)\]

**writebas (string):** produce a Cplex basis file

Write a basis file.

**writeflt (string):** produce a Cplex solution pool filter file

Write the diversity filter to a Cplex FLT file.

**writelp (string):** produce a Cplex LP file

Write a file in Cplex LP format.

**writemps (string):** produce a Cplex MPS file

Write an MPS problem file.

**writemst (string):** produce a Cplex mst file

Write a Cplex mst (containing the mip start) file.

**writeord (string):** produce a Cplex ord file
Write a Cplex ord (containing priority and branch direction information) file.

writeparam (string): produce a Cplex parameter file with all active options

Write a Cplex parameter (containing all modified Cplex options) file.

writepre (string): produce a Cplex LP/MPS/SAV file of the presolved problem

Synonym: writepremps

Write a Cplex LP, MPS, or SAV file of the presolved problem. The file extension determines the problem format. For example, writepre presolved.lp creates a file presolved.lp in Cplex LP format.

writesav (string): produce a Cplex binary problem file

Write a binary problem file.

zerohalfcuts (integer): zero-half cuts

Decides whether or not to generate zero-half cuts for the problem. The value 0, the default, specifies that the attempt to generate zero-half cuts should continue only if it seems to be helping. If the dual bound of your model does not make sufficient progress, consider setting this parameter to 2 to generate zero-half cuts more aggressively.

(default = 0)

-1 Off
0 Automatic
1 Generate zero-half cuts moderately
2 Generate zero-half cuts aggressively
1 DECIS

1.1 Introduction

DECIS is a system for solving large-scale stochastic programs, programs, which include parameters (coefficients and right-hand sides) that are not known with certainty, but are assumed to be known by their probability distribution. It employs Benders decomposition and allows using advanced Monte Carlo sampling techniques. DECIS includes a variety of solution strategies, such as solving the universe problem, the expected value problem, Monte Carlo sampling within the Benders decomposition algorithm, and Monte Carlo pre-sampling. When using Monte Carlo sampling the user has the option of employing crude Monte Carlo without variance reduction techniques, or using as variance reduction techniques importance sampling or control variates, based on either an additive or a multiplicative approximation function. Pre-sampling is limited to using crude Monte Carlo only.

For solving linear and nonlinear programs (master and subproblems arising from the decomposition) DECIS interfaces with MINOS or CPLEX. MINOS, see Murtagh and Saunders (1983) [4], is a state-of-the-art solver for large-scale linear and nonlinear programs, and CPLEX, see CPLEX Optimization, Inc. (1989–1997) [2], is one of the fastest linear programming solvers available.

For details about the DECIS system consult the DECIS User’s Guide see Infanger (1997) [3]. It includes a comprehensive mathematical description of the methods used by DECIS. In this Guide we concentrate on how to use DECIS directly from GAMS, see Brooke, A., Kendrik, D. and Meeraus, A. (1988) [1], and especially on how to model stochastic programs using the GAMS/DECIS interface. First, however, in section What DECIS Can Do we give a brief description of what DECIS can do and what solution strategies it uses. This description has been adapted from the DECIS User’s Guide. In section GAMS/DECIS we discuss in detail how to set up a stochastic problem using GAMS/DECIS and give a description of the parameter setting and outputs obtained. In Appendix A - GAMS/DECIS Illustrative Examples we show the GAMS/DECIS formulation of two illustrative examples (APL1P and APL1PC) discussed in the DECIS User’s Guide. A list of DECIS error messages are represented in Appendix B - Error Messages.

Besides the syntax explained in this document it is also possible to use the Extended Mathematical Programming (EMP) framework to define a stochastic program for GAMS/DECIS. This possibility was introduced with GAMS 23.8 and allows to solve a stochastic program not only with GAMS/DECIS but also with GAMS/DE and GAMS/Lindo. Further information about this can be found in the chapter "Stochastic Programming (SP) with EMP".

1.2 What DECIS Can Do

DECIS solves two-stage stochastic linear programs with recourse:

\[
\begin{align*}
\min \ z &= cx + \mathbb{E} f^\omega y^\omega \\
\text{s.t} \hspace{1cm} &Ax = b \\
&-B^\omega x + D^\omega y^\omega = d^\omega \\
&x, \ y^\omega \geq 0, \ \omega \in \Omega.
\end{align*}
\]

where \(x\) denotes the first-stage, \(y^\omega\) the second-stage decision variables, \(c\) represents the first-stage and \(f^\omega\) the second-stage objective coefficients, \(A, b\) represent the coefficients and right hand sides of the first-stage constraints, and \(B^\omega, D^\omega, d^\omega\) represent the parameters of the second-stage constraints, where the transition matrix \(B^\omega\) couples the two stages. In the literature \(D^\omega\) is often referred to as the technology matrix or recourse matrix. The first stage parameters are known with certainty. The second stage parameters are random parameters that assume outcomes labeled \(\omega\) with probability \(p(\omega)\), where \(\Omega\) denotes the set of all possible outcome labels.

At the time the first-stage decision \(x\) has to be made, the second-stage parameters are only known by their probability distribution of possible outcomes. Later after \(x\) is already determined, an actual outcome of the second-stage parameters will become known, and the second-stage decision \(y^\omega\) is made based on knowledge of the actual outcome \(\omega\). The objective is to find a feasible decision \(x\) that minimizes the total expected costs, the sum of first-stage costs and expected second-stage costs.
For discrete distributions of the random parameters, the stochastic linear program can be represented by the corresponding equivalent deterministic linear program:

\[
\begin{align*}
\min_{s/t} & \quad z = cx + p^1 f y^1 + p^2 f y^2 + \cdots + p^W f y^W \\
\text{s.t.} & \quad Ax - B^1 x + D y^1 = b \\
& \quad -B^2 x + D y^2 = d^2 \\
& \quad \vdots \\
& \quad -B^W x + D y^W = d^W \\
& \quad x, y^1, y^2, \ldots, y^W \geq 0,
\end{align*}
\]

which contains all possible outcomes \( \omega \in \Omega \). Note that for practical problems \( W \) is very large, e.g., a typical number could be \( 10^{20} \), and the resulting equivalent deterministic linear problem is too large to be solved directly.

In order to see the two-stage nature of the underlying decision making process the following representation is also often used:

\[
\begin{align*}
\min_{s/t} & \quad cx + E z^{\omega}(x) \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}
\]

where

\[
\begin{align*}
z^{\omega}(x) &= \min f^{\omega} y^{\omega} \\
D^{\omega} y^{\omega} &= d^{\omega} + B^{\omega} x \\
y^{\omega} &\geq 0, \quad \omega \in \Omega = \{1, 2, \ldots, W\}.
\end{align*}
\]

DECIS employs different strategies to solve two-stage stochastic linear programs. It computes an exact optimal solution to the problem or approximates the true optimal solution very closely and gives a confidence interval within which the true optimal objective lies with, say, 95% confidence.

### 1.3 Representing Uncertainty

It is favorable to represent the uncertain second-stage parameters in a structure. Using \( V = (V_1, \ldots, V_h) \) an \( h \)-dimensional independent random vector parameter that assumes outcomes \( v^{\omega} = (v_1^{\omega}, \ldots, v_h^{\omega}) \) with probability \( p^{\omega} = p(v^{\omega}) \), we represent the uncertain second-stage parameters of the problem as functions of the independent random parameter \( V \):

\[
\begin{align*}
f^{\omega} &= f(v^{\omega}), \quad B^{\omega} = B(v^{\omega}), \quad D^{\omega} = D(v^{\omega}), \quad d^{\omega} = d(v^{\omega}).
\end{align*}
\]

Each component \( V_i \) has outcomes \( v_i^{\omega_i} \), \( \omega_i \in \Omega_i \), where \( \omega_i \) labels a possible outcome of component \( i \), and \( \Omega_i \) represents the set of all possible outcomes of component \( i \). An outcome of the random vector

\[
v^{\omega} = (v_1^{\omega_1}, \ldots, v_h^{\omega_h})
\]

consists of \( h \) independent component outcomes. The set

\[
\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_h
\]

represents the crossing of sets \( \Omega_i \). Assuming each set \( \Omega_i \) contains \( W_i \) possible outcomes, \( |\Omega_i| = W_i \), the set \( \Omega \) contains \( W = \prod W_i \) elements, where \( |\Omega| = W \) represents the number of all possible outcomes of the random vector \( V \). Based on independence, the joint probability is the product.
\[ p^\omega = p_1^\omega p_2^\omega \cdots p_h^\omega. \]

Let \( \eta \) denote the vector of all second-stage random parameters, e.g., \( \eta = \text{vec}(f, B, D, d) \). The outcomes of \( \eta \) may be represented by the following general linear dependency model:

\[ \eta^\omega = \text{vec}(f^\omega, B^\omega, d^\omega, d^\omega) = Hv^\omega, \ \omega \in \Omega \]

where \( H \) is a matrix of suitable dimensions. DECIS can solve problems with such general linear dependency models.

### 1.4 Solving the Universe Problem

We refer to the universe problem if we consider all possible outcomes \( \omega \in \Omega \) and solve the corresponding problem exactly. This is not always possible, because there may be too many possible realizations \( \omega \in \Omega \). For solving the problem DECIS employs Benders decomposition, splitting the problem into a master problem, corresponding to the first-stage decision, and into subproblems, one for each \( \omega \in \Omega \), corresponding to the second-stage decision. The details of the algorithm and techniques used for solving the universe problem are discussed in The DECIS User’s Manual.

Solving the universe problem is referred to as strategy 4. Use this strategy only if the number of universe scenarios is reasonably small. There is a maximum number of universe scenarios DECIS can handle, which depends on your particular resources.

### 1.5 Solving the Expected Value Problem

The expected value problem results from replacing the stochastic parameters by their expectation. It is a linear program that can also easily be solved by employing a solver directly. Solving the expected value problem may be useful by itself (for example as a benchmark to compare the solution obtained from solving the stochastic problem), and it also may yield a good starting solution for solving the stochastic problem. DECIS solves the expected value problem using Benders decomposition. The details of generating the expected value problem and the algorithm used for solving it are discussed in the DECIS User’s Manual. To solve the expected value problem choose strategy 1.

### 1.6 Using Monte Carlo Sampling

As noted above, for many practical problems it is impossible to obtain the universe solution, because the number of possible realizations \( |\Omega| \) is way too large. The power of DECIS lies in its ability to compute excellent approximate solutions by employing Monte Carlo sampling techniques. Instead of computing the expected cost and the coefficients and the right-hand sides of the Benders cuts exactly (as it is done when solving the universe problem), DECIS, when using Monte Carlo sampling, estimates the quantities in each iteration using an independent sample drawn from the distribution of the random parameters. In addition to using crude Monte Carlo, DECIS uses importance sampling or control variates as variance reduction techniques.

The details of the algorithm and the different techniques used are described in the DECIS User’s Manual. You can choose crude Monte Carlo, referred to as strategy 6, Monte Carlo importance sampling, referred to as strategy 2, or control variates, referred to as strategy 10. Both Monte Carlo importance sampling and control variates have been shown for many problems to give a better approximation compared to employing crude Monte Carlo sampling.

When using Monte Carlo sampling DECIS computes a close approximation to the true solution of the problem, and estimates a close approximation of the true optimal objective value. It also computes a confidence interval within which the true optimal objective of the problem lies, say with 95% confidence. The confidence interval is based on rigorous statistical theory. An outline of how the confidence interval is computed is given in the DECIS User’s Manual. The size of the confidence interval depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for the estimation. You can expect the confidence interval to be very small, especially when you employ importance sampling or control variates as a variance reduction technique.

When employing Monte Carlo sampling techniques you have to choose a sample size (set in the parameter file). Clearly, the larger the sample size the better will be the approximate solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Setting the sample size too small may lead to bias in the estimation of the confidence interval, therefore the sample size should be at least 30.
1.7 Monte Carlo Pre-sampling

We refer to pre-sampling when we first take a random sample from the distribution of the random parameters and then generate the approximate stochastic problem defined by the sample. The obtained approximate problem is then solved exactly using decomposition. This is in contrast to the way we used Monte Carlo sampling in the previous section, where we used Monte Carlo sampling in each iteration of the decomposition.

The details of the techniques used for pre-sampling are discussed in the DECIS User’s Manual. DECIS computes the exact solution of the sampled problem using decomposition. This solution is an approximate solution of the original stochastic problem. Besides this approximate solution, DECIS computes an estimate of the expected cost corresponding to this approximate solution and a confidence interval within which the true optimal objective of the original stochastic problem lies with, say, 95% confidence. The confidence interval is based on statistical theory, its size depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for generating the approximate problem. In conjunction with pre-sampling no variance reduction techniques are currently implemented.

Using Monte Carlo pre-sampling you have to choose a sample size. Clearly, the larger the sample size you choose, the better will be the solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Again, setting the sample size as too small may lead to a bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

For using Monte Carlo pre-sampling choose strategy 8.

1.8 Regularized Decomposition

When solving practical problems, the number of Benders iterations can be quite large. In order to control the decomposition, with the hope to reduce the iteration count and the solution time, DECIS makes use of regularization. When employing regularization, an additional quadratic term is added to the objective of the master problem, representing the square of the distance between the best solution found so far (the incumbent solution) and the variable $x$. Using this term, DECIS controls the distance of solutions in different decomposition iterations.

For enabling regularization you have to set the corresponding parameter. You also have to choose the value of the constant rho in the regularization term. The default is regularization disabled. Details of how DECIS carries out regularization are represented in the DECIS User’s Manual.

Regularization is only implemented when using MINOS as the optimizer for solving subproblems. Regularization has proven to be helpful for problems that need a large number of Benders iteration when solved without regularization. Problems that need only a small number of Benders iterations without regularization are not expected to improve much with regularization, and may need even more iterations with regularization than without.

2 GAMS/DECIS

GAMS stands for General Algebraic Modeling Language, and is one of the most widely used modeling languages. Using DECIS directly from GAMS spares you from worrying about all the details of the input formats. It makes the problem formulation much easier but still gives you almost all the flexibility of using DECIS directly.

The link from GAMS to DECIS has been designed in such a way that almost no extensions to the GAMS modeling language were necessary for carrying out the formulation and solution of stochastic programs. In a next release of GAMS, however, additions to the language are planned that will allow you to model stochastic programs in an even more elegant way.

2.1 Setting up a Stochastic Program Using GAMS/DECIS

The interface from GAMS to DECIS supports the formulation and solution of stochastic linear programs. DECIS solves them using two-stage decomposition. The GAMS/DECIS interface resembles closely the structure of the SMPS (stochastic mathematical programming interface) discussed in the DECIS User’s Manual. The specification of a stochastic problem using GAMS/DECIS uses the following components:

- the deterministic (core) model,
• item the specification of the decision stages,
• item the specification of the random parameters, and
• item setting DECIS to be the optimizer to be used.

2.2 Starting with the Deterministic Model

The core model is the deterministic linear program where all random parameters are replaced by their mean or by a particular realization. One could also see it as a GAMS model model without any randomness. It could be a deterministic model that you have, which you intend to expand to a stochastic one. Using DECIS with GAMS allows you to easily extend a deterministic linear programming model to a stochastic one. For example, the following GAMS model represents the a deterministic version of the electric power expansion planning illustrative example discussed in Infanger (1994).

* APL1P test model
* Dr. Gerd Infanger, November 1997
* Deterministic Program

set g generators / g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter ccmin(g) min capacity / g1 1000, g2 1000 /;
parameter ccmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

table f(g,dl) operating cost
   h   m   l
  g1  4.3  2.0  0.5
  g2  8.7  4.0  1.0;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost total cost;
positive variable x(g) capacity of generators;
positive variable y(g, dl) operating level;
positive variable s(dl) unserved demand;

equations
cost total cost
cmin(g) minimum capacity
cmax(g) maximum capacity
omax(g) maximum operating level
demand(dl) satisfy demand;

cost .. tcost =e= sum(g, c(g)*x(g))
+ sum(g, sum(dl, f(g,dl)*y(g,dl)))
+ sum(dl,us(dl)*s(dl));
cmin(g) .. x(g) =g= ccmin(g);
cmax(g) .. x(g) =l= ccmax(g);
omax(g) .. sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apilip /all/;
option lp=minos5;
solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;

### 2.3 Setting the Decision Stages

Next in order to extend a deterministic model to a stochastic one you must specify the decision stages. DECIS solves stochastic programs by two-stage decomposition. Accordingly, you must specify which variables belong to the first stage and which to the second stage, as well as which constraints are first-stage constraints and which are second-stage constraints. First stage constraints involve only first-stage variables, second-stage constraints involve both first- and second-stage variables. You must specify the stage of a variable or a constraint by setting the stage suffix ".STAGE" to either one or two depending on if it is a first or second stage variable or constraint. For example, expanding the illustrative model above by

* setting decision stages
  x.stage(g) = 1;
y.stage(g, dl) = 2;
s.stage(dl) = 2;
cmin.stage(g) = 1;
cmax.stage(g) = 1;
onmax.stage(g) = 2;
demand.stage(dl) = 2;

would make x(g) first-stage variables, y(g, dl) and s(dl) second-stage variables, cmin(g) and cmax(g) first-stage constraints, and onmax(g) and demand(g) second-stage constraints. The objective is treated separately, you don’t need to set the stage suffix for the objective variable and objective equation.

It is noted that the use of the .stage variable and equation suffix causes the GAMS scaling facility through the .scale suffices to be unavailable. Stochastic models have to be scaled manually.

### 2.4 Specifying the Stochastic Model

DECIS supports any linear dependency model, i.e., the outcomes of an uncertain parameter in the linear program are a linear function of a number of independent random parameter outcomes. DECIS considers only discrete distributions, you must approximate any continuous distributions by discrete ones. The number of possible realizations of the discrete random parameters determines the accuracy of the approximation. A special case of a linear dependency model arises when you have only independent random parameters in your model. In this case the independent random parameters are mapped one to one into the random parameters of the stochastic program. We will present the independent case first and then expand to the case with linear dependency. According to setting up a linear dependency model we present the formulation in GAMS by first defining independent random parameters and then defining the distributions of the uncertain parameters in your model.

#### Specifying Independent Random Parameters

There are of course many different ways you can set up independent random parameters in GAMS. In the following we show one possible way that is generic and thus can be adapted for different models. The set-up uses the set stock for labeling outcome named "out" and probability named "pro" of each independent random parameter. In the following we show how to define an independent random parameter, say, v1. The formulation uses the set omega1 as driving set, where the set contains one element for each possible realization the random parameter can assume. For example, the set omega1 has four elements according to a discrete distribution of four possible outcomes. The distribution of the random parameter is defined as the
parameter \( v_1 \), a two-dimensional array of outcomes "out" and corresponding probability "pro" for each of the possible realizations of the set \( \omega_1 \), "o11", "o12", "o13", and "o14". For example, the random parameter \( v_1 \) has outcomes of \(-1.0, -0.9, -0.5, -0.1\) with probabilities \(0.2, 0.3, 0.4, 0.1\), respectively. Instead of using assignment statements for inputting the different realizations and corresponding probabilities you could also use the table statement. You could also the table statement would work as well. Always make sure that the sum of the probabilities of each independent random parameter adds to one.

* defining independent stochastic parameters
set stoch /out, pro /
set omega1 / o11, o12, o13, o14 /;

table v1(stoch, omega1)
  o11 o12 o13 o14
  out -1.0 -0.9 -0.5 -0.1
  pro 0.2 0.3 0.4 0.1
;

Random parameter \( v_1 \) is the first out of five independent random parameters of the illustrative model APL1P, where the first two represent the independent availabilities of the generators \( g_1 \) and \( g_2 \) and the latter three represent the independent demands of the demand levels \( h \), \( m \), and \( l \). We also represent the definitions of the remaining four independent random parameters. Note that random parameters \( v_3 \), \( v_4 \), and \( v_5 \) are identically distributed.

set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
  o21 o22 o23 o24 o25
  out -1.0 -0.9 -0.7 -0.1 -0.0
  pro 0.1 0.2 0.5 0.1 0.1
;

set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
  o11 o12 o13 o14
  out 900 1000 1100 1200
  pro 0.15 0.45 0.25 0.15
;

set omega4 / o41, o42, o43, o44 /;
table v4(stoch, omega1)
  o11 o12 o13 o14
  out 900 1000 1100 1200
  pro 0.15 0.45 0.25 0.15
;

set omega5 / o51, o52, o53, o54 /;
table v5(stoch, omega1)
  o11 o12 o13 o14
  out 900 1000 1100 1200
  pro 0.15 0.45 0.25 0.15
;

Defining the Distributions of the Uncertain Parameters in the Model

Having defined the independent stochastic parameters (you may copy the setup above and adapt it for your model), we next define the stochastic parameters in the GAMS model. The stochastic parameters of the model are defined by writing a file,
the GAMS stochastic file, using the put facility of GAMS. The GAMS stochastic file resembles closely the stochastic file of the SMPS input format. The main difference is that we use the row, column, bounds, and right hand side names of the GAMS model and that we can write it in free format.

Independent Stochastic Parameters

First we describe the case where all stochastic parameters in the model are independent, see below the representation of the stochastic parameters for the illustrative example APL1P, which has five independent stochastic parameters.

First define the GAMS stochastic file "MODEL.STG" (only the exact name in uppercase letters is supported) and set up GAMS to write to it. This is done by the first two statements. You may want to consult the GAMS manual for how to use put for writing files. The next statement "INDEP DISCRETE" indicates that a section of independent stochastic parameters follows. Then we write all possible outcomes and corresponding probabilities for each stochastic parameter best by using a loop statement. Of course one could also write each line separately, but this would not look nicely. Writing a "*" between the definitions of the independent stochastic parameters is merely for optical reasons and can be omitted.

* defining distributions (writing file MODEL.STG)
file stg /MODEL.STG/;
put stg;
put "INDEP DISCRETE" /;
  loop(omega1,
    put "x g1 omax g1 ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
  );
  put "*" /;
  loop(omega2,
    put "x g2 omax g2 ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
  );
  put "*" /;
  loop(omega3,
    put "RHS demand h ", v3("out", omega3), " period2 ", v3("pro", omega3) /;
  );
  put "*" /;
  loop(omega4,
    put "RHS demand m ", v4("out", omega4), " period2 ", v4("pro", omega4) /;
  );
  put "*" /;
  loop(omega5,
    put "RHS demand l ", v5("out", omega5), " period2 ", v5("pro", omega5) /;
  );
putclose stg;

In the example APL1P the first stochastic parameter is the availability of generator g1. In the model the parameter appears as the coefficient of variable x(g1) in equation omax(g1). The definition using the put statement first gives the stochastic parameter as the intersection of variable x(g1) with equation omax(g1), but without having to type the braces, thus x g1 omax g1, then the outcome v1("out", omega1) and the probability v1("pro", omega1) separated by "period2". The different elements of the statement must be separated by blanks. Since the outcomes and probabilities of the first stochastic parameters are driven by the set omega1 we loop over all elements of the set omega1. We continue and define all possible outcomes for each of the five independent stochastic parameters.

In the example of independent stochastic parameters, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

INDEP DISCRETE
  x g1 omax g1 -1.00 period2 0.20
  x g1 omax g1 -0.90 period2 0.30
x g1 omax g1  -0.50 period2  0.40
x g1 omax g1  -0.10 period2  0.10
*

x g2 omax g2  -1.00 period2  0.10
x g2 omax g2  -0.90 period2  0.20
x g2 omax g2  -0.70 period2  0.50
x g2 omax g2  -0.10 period2  0.10
x g2 omax g2  0.00 period2  0.10
*

RHS demand h  900.00 period2  0.15
RHS demand h  1000.00 period2  0.45
RHS demand h  1100.00 period2  0.25
RHS demand h  1200.00 period2  0.15
*

RHS demand m  900.00 period2  0.15
RHS demand m  1000.00 period2  0.45
RHS demand m  1100.00 period2  0.25
RHS demand m  1200.00 period2  0.15
*

RHS demand l  900.00 period2  0.15
RHS demand l  1000.00 period2  0.45
RHS demand l  1100.00 period2  0.25
RHS demand l  1200.00 period2  0.15

For defining stochastic parameters in the right-hand side of the model use the keyword RHS as the column name, and the equation name of the equation which right-hand side is uncertain, see for example the specification of the uncertain demands RHS demand h, RHS demand m, and RHS demand l. For defining uncertain bound parameters you would use the keywords UP, LO, or FX, the string bnd, and the variable name of the variable, which upper, lower, or fixed bound is uncertain.

Note all the keywords for the definitions are in capital letters, i.e., ”INDEP DISCRETE”, “RHS”, and not represented in the example ”UP”, ”LO”, and ”FX”.

It is noted that in GAMS equations, variables may appear in the right-hand side, e.g. ”EQ .. X+1 =L= 2*Y”. When the coefficient 2 is a random variable, we need to be aware that GAMS will generate the following LP row X - 2*Y =L= -1.

Suppose the probability distribution of this random variable is given by:

\[
\text{set s scenario /pessimistic, average, optimistic/;}
\]
\[
\text{parameter outcome(s) / pessimistic 1.5}
\quad\text{average 2.0}
\quad\text{optimistic 2.3 /;}
\]
\[
\text{parameter prob(s) / pessimistic 0.2}
\quad\text{average 0.6}
\quad\text{optimistic 0.2 /;}
\]

then the correct way of generating the entries in the stochastic file would be:

\[
\text{loop(s,}
\quad\text{put "Y EQ ",(-outcome(s))," PERIOD2 ",prob(s)/;}
\);}

Note the negation of the outcome parameter. Also note that expressions in a PUT statement have to be surrounded by parentheses. GAMS reports in the row listing section of the listing file how equations are generated. You are encouraged to inspect the row listing how coefficients appear in a generated LP row.
Dependent Stochastic Parameters

Next we describe the case of general linear dependency of the stochastic parameters in the model, see below the representation of the stochastic parameters for the illustrative example APL1PCA, which has three dependent stochastic demands driven by two independent stochastic random parameters. First we give the definition of the two independent stochastic parameters, which in the example happen to have two outcomes each.

* defining independent stochastic parameters
set stoch / out, pro /;
set omega1 / o11, o12 /;
table v1(stoch, omega1)
    o11  o12
    out  2.1  1.0
    pro  0.5  0.5 ;
set omega2 / o21, o22 /;
table v2(stoch, omega2)
    o21  o22
    out  2.0  1.0
    pro  0.2  0.8 ;

We next define the parameters of the transition matrix from the independent stochastic parameters to the dependent stochastic parameters of the model. We do this by defining two parameter vectors, where the vector $h m 1$ gives the coefficients of the independent random parameter $v 1$ in each of the three demand levels and the vector $h m 2$ gives the coefficients of the independent random parameter $v 2$ in each of the three demand levels.

parameter hm1(dl) / h 300., m 400., l 200. /;
parameter hm2(dl) / h 100., m 150., l 300. /;

Again first define the GAMS stochastic file "MODEL.STG" and set GAMS to write to it. The statement \textit{BLOCKS DISCRETE} indicates that a section of linear dependent stochastic parameters follows.

* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
    put "BL v1 period2 ", v1("pro", omega1) /;
    loop(dl,
        h1 = hm1(dl) * v1("out", omega1);
        put "RHS demand ", dl.tl:1, " ", h1 /;
    );
);
loop(omega2,
    put " BL v2 period2 ", v2("pro", omega2) /;
    loop(dl,
        h1 = hm2(dl) * v2("out", omega2);
        put "RHS demand ", dl.tl:1, " ", h1 /;
    );
);
putclose stg;
Dependent stochastic parameters are defined as functions of independent random parameters. The keyword \textit{BL} labels a possible realization of an independent random parameter. The name besides the \textit{BL} keyword is used to distinguish between different outcomes of the same independent random parameter or a different one. While you could use any unique names for the independent random parameters, it appears natural to use the names you have already defined above, e.g., \textit{v1} and \textit{v2}. For each realization of each independent random parameter define the outcome of every dependent random parameter (as a function of the independent one). If a dependent random parameter in the GAMS model depends on two or more different independent random parameter the contributions of each of the independent parameters are added. We are therefore in the position to model any linear dependency model. (Note that the class of models that can be accommodated here is more general than linear. The functions, with which an independent random variable contributes to the dependent random variables can be any ones in one argument. As a general rule, any stochastic model that can be estimated by linear regression is supported by GAMS/DECIS.)

Define each independent random parameter outcome and the probability associated with it. For example, the statement starting with \textit{BL v1 period2} indicates that an outcome of (independent random parameter) \textit{v1} is being defined. The name \textit{period2} indicates that it is a second-stage random parameter, and \textit{v1("pro", omega1)} gives the probability associated with this outcome. Next list all random parameters dependent on the independent random parameter outcome just defined. Define the dependent stochastic parameter coefficients by the GAMS variable name and equation name, or “RHS” and variable name, together with the value of the parameter associated with this realization. In the example, we have three dependent demands. Using the scalar \textit{h1} for intermediately storing the results of the calculation, looping over the different demand levels \textit{dl} we calculate \textit{h1 = hm1(dl) * v1("out", omega1)} and define the dependent random parameters as the right-hand sides of equation demand(\textit{dl}).

When defining an independent random parameter outcome, if the block name is the same as the previous one (e.g., when \textit{BL v1} appears the second time), a different outcome of the same independent random parameter is being defined, while a different block name (e.g., when \textit{BL v2} appears the first time) indicates that the first outcome of a different independent random parameter is being defined. You must ensure that the probabilities of the different outcomes of each of the independent random parameters add up to one. The loop over all elements of \textit{omega1} defines all realizations of the independent random parameter \textit{v1} and the loop over all elements of \textit{omega2} defines all realizations of the independent random parameter \textit{v2}.

Note for the first realization of an independent random parameter, you must define all dependent parameters and their realizations. The values entered serve as a base case. For any other realization of an independent random parameter you only need to define the dependent parameters that have different coefficients than have been defined in the base case. For those not defined in a particular realization, their values of the base case are automatically added.

In the example of dependent stochastic parameters above, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```
BLOCKS DISCRETE
BL v1 period2 0.50
RHS demand h 630.00
RHS demand m 840.00
RHS demand l 420.00
BL v1 period2 0.50
RHS demand h 300.00
RHS demand m 400.00
RHS demand l 200.00
BL v2 period2 0.20
RHS demand h 200.00
RHS demand m 300.00
RHS demand l 600.00
BL v2 period2 0.80
RHS demand h 100.00
RHS demand m 150.00
RHS demand l 300.00
```

Again all the keywords for the definitions are in capital letters, i.e., "BLOCKS DISCRETE", "BL", "RHS", and not represented in the example "UP", "LO", and "FX".
Note that you can only define random parameter coefficients that are nonzero in your GAMS model. When setting up the
deterministic core model put a nonzero entry as a placeholder for any coefficient that you wish to specify as a stochastic
parameter. Specifying a random parameter at the location of a zero coefficient in the GAMS model causes DECIS to terminate
with an error message.

2.5 Setting DECIS as the Optimizer

After having finished the stochastic definitions you must set DECIS as the optimizer. This is done by issuing the following
statements:

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
ap1lp.optfile = 1;

The statement option lp = decism sets DECIS with the MINOS LP engine as the optimizer to be used for solving the
stochastic problem. Note that if you do not use DECIS, but instead use any other linear programming optimizer, your GAMS
model will still run and optimize the deterministic core model that you have specified. The statement ap1lp.optfile = 1 forces
GAMS to process the file DECIS.OPT, in which you may define any DECIS parameters.

Setting Parameter Options in the GAMS Model

The options iteration limit and resource limit can be set directly in your GAMS model file. For example, the following
statements

option iterlim = 1000;
option reslim = 6000;

constrain the number of decomposition iterations to be less than or equal to 1000, and the elapsed time for running DECIS to
be less than or equal to 6000 seconds or 100 minutes.

Setting Parameters in the DECIS Options File

In the DECIS options file DECIS.OPT you can specify parameters regarding the solution algorithm used and control the
output of the DECIS program. There is a record for each parameter you want to specify. Each record consists of the value of
the parameter you want to specify and the keyword identifying the parameter, separated by a blank character or a comma.
You may specify parameters with the following keywords: istrat, nsamples, nzrows, iwrite, ibug, iscratch, ireg, rho, tolben,
and tolw in any order. Each keyword can be specified in lower case or upper case text in the format (A10). Since DECIS
reads the records in free format you don’t have to worry about the format, but some computers require that the text is inputted
in quotes. Parameters that are not specified in the parameter file automatically assume their default values. See details of
these parameters from Section Description of GAMS/DECIS Options.

Example

In the following example the parameters istrat = 7, nsamples = 200, and nzrows = 200 are specified. All other parameters are
set at their default values. DECIS first solves the expected value problem and then the stochastic problem using crude Monte
Carlo sampling with a sample size of nsamples = 200. DECIS reserves space for a maximum of nzrows = 50 cuts.

7 "ISTRAT"
200 "NSAMPLES"
50 "NZROWS"
Setting MINOS Parameters in the MINOS Specification File

When you use MINOS as the optimizer for solving the master and the subproblems, you must specify optimization parameters in the MINOS specification file "MINOS.SPC". Each record of the file corresponds to the specification of one parameter and consists of a keyword and the value of the parameter in free format. Records having a "∗" as their first character are considered as comment lines and are not further processed. For a detailed description of these parameters, see the MINOS Users’ Guide (Murtagh and Saunders (1983) [4]. The following parameters should be specified with some consideration:

- **AIJ TOLERANCE** — Specifies the nonzero tolerance for constraint matrix of the problem. Matrix elements $a_{ij}$ that have a value for which $|a_{ij}|$ is less than "AIJ TOLERANCE" are considered by MINOS as zero and are automatically eliminated from the problem. It is wise to specify "AIJ TOLERANCE 0.0 ."

- **SCALE** — Specifies MINOS to scale the problem ("SCALE YES") or not ("SCALE NO"). It is wise to specify "SCALE NO".

- **ROWS** — Specifies the number of rows in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ROWS" should be specified as the number of constraints in the core problem or greater.

- **COLUMNS** — Specifies the number of columns in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "COLUMNS" should be specified as the number of variables in the core problem or greater.

- **ELEMENTS** — Specifies the number of nonzero matrix coefficients in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ELEMENTS" should be specified as the number of nonzero matrix coefficients in the core problem or greater.

**Example**

The following example represents typical specifications for running DECIS with MINOS as the optimizer.

```
BEGIN SPECS
PRINT LEVEL 1
LOG FREQUENCY 10
SUMMARY FREQUENCY 10
MPS FILE 12
ROWS 20000
COLUMNS 50000
ELEMENTS 100000
ITERATIONS LIMIT 30000
*
FACTORIZATION FREQUENCY 100
AIJ TOLERANCE 0.0
*
SCALE NO
END OF SPECS
```

Setting CPLEX Parameters Using System Environment Variables

When you use CPLEX as the optimizer for solving the master and the subproblems, optimization parameters must be specified through system environment variables. You can specify the parameters "CPLEXLICDIR", "SCALELP", "NOPRESOLVE", "ITERLOG", "OPTIMALITYTOL", "FEASIBILITYTOL", and "DUALSIMPLEX".

- **CPLEXLICDIR** — Contains the path to the CPLEX license directory. For example, on an Unix system with the CPLEX license directory in /usr/users/cplex/cplexlicdir you issue the command `setenv CPLEXLICDIR /usr/users/cplex/cplexlicdir`.


• SCALELP — Specifies CPLEX to scale the master and subproblems before solving them. If the environment variable is not set no scaling is used. Setting the environment variable, e.g., by issuing the command `setenv SCALELP yes`, scaling is switched on.

• NOPRESOLVE — Allows to switch off CPLEX’s presolver. If the environment variable is not set, presolve will be used. Setting the environment variable, e.g., by setting `setenv NOPRESOLVE yes`, no presolve will be used.

• ITERLOG — Specifies the iteration log of the CPLEX iterations to be printed to the file "MODEL.CPX". If you do not set the environment variable no iteration log will be printed. Setting the environment variable, e.g., by setting `setenv ITERLOG yes`, the CPLEX iteration log is printed.

• OPTIMALITYTOL — Specifies the optimality tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting `setenv OPTIMALITYTOL 1.0E-7` sets the CPLEX optimality tolerance to 0.0000001.

• FEASIBILITYTOL — Specifies the feasibility tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting `setenv FEASIBILITYTOL 1.0E-7` sets the CPLEX optimality tolerance to 0.0000001.

• DUALSIMPLEX — Specifies the dual simplex algorithm of CPLEX to be used. If the environment variable is not set the primal simplex algorithm will be used. This is the default and works beautifully for most problems. If the environment variable is set, e.g., by setting `setenv DUALSIMPLEX yes`, CPLEX uses the dual simplex algorithm for solving both master and subproblems.

2.6 GAMS/DECIS Output

After successfully having solved a problem, DECIS returns the objective, the optimal primal and optimal dual solution, the status of variables (if basic or not), and the status of equations (if binding or not) to GAMS. In the case of first-stage variables and equations you have all information in GAMS available as if you used any other solver, just instead of obtaining the optimal values for deterministic core problem you actually obtained the optimal values for the stochastic problem. However, for second-stage variables and constraints the expected values of the optimal primal and optimal dual solution are reported. This saves space and is useful for the calculation of risk measures. However, the information as to what the optimal primal and dual solutions were in the different scenarios of the stochastic programs is not reported back to GAMS. In a next release of the GAMS/DECIS interface the GAMS language is planned to be extended to being able to handle the scenario second-stage optimal primal and dual values at least for selected variables and equations.

While running DECIS outputs important information about the progress of the execution to your computer screen. After successfully having solved a problem, DECIS also outputs its optimal solution into the solution output file "MODEL.SOL". The debug output file "MODEL.SCR" contains important information about the optimization run, and the optimizer output files "MODEL.MO" (when using DECIS with MINOS) or "MODEL.CPX" (when using DECIS with CPLEX) contain solution output from the optimizer used. In the DECIS User’s Guide you find a detailed discussion of how to interpret the screen output, the solution report and the information in the output files.

The Screen Output

The output to the screen allows you to observe the progress in the execution of a DECIS run. After the program logo and the copyright statement, you see four columns of output being written to the screen as long as the program proceeds. The first column (from left to right) represents the iteration count, the second column the lower bound (the optimal objective of the master problem), the third column the best upper bound (exact value or estimate of the total expected cost of the best solution found so far), and the fourth column the current upper bound (exact value or estimate of the total expected cost of current solution). After successful completion, DECIS quits with "Normal Exit", otherwise, if an error has been encountered, the programs stops with the message "Error Exit".

Example

When solving the illustrative example APL1P using strategy 5, we obtain the following report on the screen:
### The Solution Output File

The solution output file contains the solution report from the DECIS run. Its name is "MODEL.SOL". The file contains the best objective function value found, the corresponding values of the first-stage variables, the corresponding optimal second-stage cost, and a lower and an upper bound on the optimal objective of the problem. In addition, the number of universe scenarios and the settings for the stopping tolerance are reported. In the case of using a deterministic strategy for solving the problem, exact values are reported. When using Monte Carlo sampling, estimated values, their variances, and the sample size used for the estimation are reported. Instead of exact upper and lower bounds, probabilistic upper and lower bounds, and a 95% confidence interval, within which the true optimal solution lies with 95% confidence, are reported. A detailed description of the solution output file can be found in the DECIS User’s Guide.

### The Debug Output File

The debug output file contains the standard output of a run of DECIS containing important information about the problem, its parameters, and its solution. It also contains any error messages that may occur during a run of DECIS. In the case that DECIS does not complete a run successfully, the cause of the trouble can usually be located using the information in the debug output file. If the standard output does not give enough information you can set the debug parameter ibug in the parameter input file to a higher value and obtain additional debug output. A detailed description of the debug output file can be found in the DECIS User’s Guide.

### The Optimizer Output Files

The optimizer output file "MODEL.MO" contains all the output from MINOS when called as a subroutine by DECIS. You can specify what degree of detail should be outputted by setting the appropriate "PRINT LEVEL" in the MINOS specification file. The optimizer output file "MODEL.CPX" reports messages and the iteration log (if switchwd on using the environment

```plaintext
<table>
<thead>
<tr>
<th>iter</th>
<th>lower</th>
<th>best upper</th>
<th>current upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.9935E+06</td>
<td>0.2590E+05</td>
<td>0.2590E+05</td>
</tr>
<tr>
<td>1</td>
<td>-0.4626E+06</td>
<td>0.2590E+05</td>
<td>0.5487E+06</td>
</tr>
<tr>
<td>3</td>
<td>0.2170E+05</td>
<td>0.2590E+05</td>
<td>0.2697E+05</td>
</tr>
<tr>
<td>4</td>
<td>0.2368E+05</td>
<td>0.2384E+05</td>
<td>0.2384E+05</td>
</tr>
<tr>
<td>5</td>
<td>0.2370E+05</td>
<td>0.2384E+05</td>
<td>0.2401E+05</td>
</tr>
<tr>
<td>6</td>
<td>0.2370E+05</td>
<td>0.2370E+05</td>
<td>0.2370E+05</td>
</tr>
<tr>
<td>7</td>
<td>0.2403E+05</td>
<td>0.2470E+05</td>
<td>0.2470E+05</td>
</tr>
<tr>
<td>8</td>
<td>0.2433E+05</td>
<td>0.2470E+05</td>
<td>0.2694E+05</td>
</tr>
<tr>
<td>9</td>
<td>0.2441E+05</td>
<td>0.2470E+05</td>
<td>0.2602E+05</td>
</tr>
<tr>
<td>10</td>
<td>0.2453E+05</td>
<td>0.2470E+05</td>
<td>0.2499E+05</td>
</tr>
<tr>
<td>11</td>
<td>0.2455E+05</td>
<td>0.2470E+05</td>
<td>0.2483E+05</td>
</tr>
<tr>
<td>12</td>
<td>0.2461E+05</td>
<td>0.2467E+05</td>
<td>0.2467E+05</td>
</tr>
<tr>
<td>13</td>
<td>0.2461E+05</td>
<td>0.2467E+05</td>
<td>0.2469E+05</td>
</tr>
<tr>
<td>14</td>
<td>0.2461E+05</td>
<td>0.2465E+05</td>
<td>0.2465E+05</td>
</tr>
<tr>
<td>15</td>
<td>0.2463E+05</td>
<td>0.2465E+05</td>
<td>0.2467E+05</td>
</tr>
<tr>
<td>16</td>
<td>0.2463E+05</td>
<td>0.2465E+05</td>
<td>0.2465E+05</td>
</tr>
<tr>
<td>17</td>
<td>0.2464E+05</td>
<td>0.2465E+05</td>
<td>0.2465E+05</td>
</tr>
<tr>
<td>18</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
</tr>
<tr>
<td>19</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
</tr>
<tr>
<td>20</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
</tr>
<tr>
<td>21</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
</tr>
<tr>
<td>22</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
<td>0.2464E+05</td>
</tr>
</tbody>
</table>
```

Normal Exit
variable) from CPLEX when solving master and sub problems.

3 Description of GAMS/DECIS Options

3.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVSubSolver</td>
<td>Subsolver to run on expected value problem</td>
<td></td>
</tr>
<tr>
<td>EVSubSolverOpt</td>
<td>Optfile value to pass to the subsolver for expected value problem</td>
<td>1</td>
</tr>
<tr>
<td>FileName</td>
<td>Filename of generated scalar GAMS model</td>
<td>decis.gms</td>
</tr>
<tr>
<td>ScenarioLog</td>
<td>Select log option for scenario solves</td>
<td>0</td>
</tr>
<tr>
<td>SolveEVProb</td>
<td>Solve and report the expected value solution</td>
<td>0</td>
</tr>
<tr>
<td>SolveScenarios</td>
<td>Maximum number of scenarios solved</td>
<td>100</td>
</tr>
<tr>
<td>SubSolvePar</td>
<td>User defined GAMS parameters for subsolve</td>
<td></td>
</tr>
<tr>
<td>SubSolver</td>
<td>Subsolver to run</td>
<td>decisc</td>
</tr>
<tr>
<td>Terminate</td>
<td>Generate the GAMS source code of the reformulated Model</td>
<td>0</td>
</tr>
</tbody>
</table>

3.2 DECIS Solver Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBug</td>
<td>Debug output</td>
<td>0</td>
</tr>
<tr>
<td>IReg</td>
<td>Indicator for regularized decomposition - MINOS only</td>
<td>0</td>
</tr>
<tr>
<td>IScratch</td>
<td>Internal unit number for output and debug</td>
<td>17</td>
</tr>
<tr>
<td>IStrat</td>
<td>Defines the solution strategy used</td>
<td>3</td>
</tr>
<tr>
<td>IWrite</td>
<td>Subproblem output</td>
<td>0</td>
</tr>
<tr>
<td>NSamples</td>
<td>Sample size used for the estimation</td>
<td>100</td>
</tr>
<tr>
<td>NZRows</td>
<td>Number of rows reserved for cuts in the master problem</td>
<td>100</td>
</tr>
<tr>
<td>Rho</td>
<td>rho parameter of regularization term in the objective function</td>
<td>1000</td>
</tr>
<tr>
<td>TolBen</td>
<td>Tolerance for stopping the decomposition algorithm</td>
<td>1e-7</td>
</tr>
<tr>
<td>TolW</td>
<td>tolerance when writing debug solution output</td>
<td>1e-9</td>
</tr>
</tbody>
</table>

**EVSubSolver** (*string*): Subsolver to run on expected value problem

**EVSubSolverOpt** (*integer*): Optfile value to pass to the subsolver for expected value problem

  (default = 1)

**FileName** (*string*): Filename of generated scalar GAMS model

  (default = decis.gms)

**IBug** (*integer*): Debug output

  (default = 0)

  0 DECIS does not write any debug output

  1 Solution of the master problem on each iteration

  2 Value 1 plus scenario index and the optimal objective value for each subproblem solved

  3 Value 2 plus information regarding importance sampling

  4 Value 3 plus optimal dual variables of the cuts
5 Value 4 plus coefficients and the right-hand side of the cuts
6 Value 5 plus dump of the master problem and the subproblem in MPS format

**IReg (integer):** Indicator for regularized decomposition - MINOS only
(default = 0)

**IScratch (integer):** Internal unit number for output and debug
(default = 17)

**IStrat (integer):** Defines the solution strategy used
(default = 3)

1 Solves the expected value problem
2 Solves the stochastic problem using Monte Carlo importance sampling
3 Refers to istrat = 1 plus istrat = 2
4 Solves the stochastic universe problem
5 Refers to istrat = 1 plus istrat = 4
6 Solves the stochastic problem using crude Monte Carlo sampling
7 Refers to istrat = 1 plus istrat = 6
8 Solves the stochastic problem using Monte Carlo pre-sampling
9 Refers to istrat = 1 plus istrat = 8
10 Solves the stochastic problem using control variates
11 Refers to istrat = 1 plus istrat = 10

**IWrite (integer):** Subproblem output
(default = 0)

0 No optimizer output is written
1 Optimizer output is written to the file

**NSamples (integer):** Sample size used for the estimation
(default = 100)

**NZRows (integer):** Number of rows reserved for cuts in the master problem
(default = 100)

**Rho (real):** rho parameter of regularization term in the objective function
(default = 1000)

**ScenarioLog (integer):** Select log option for scenario solves
(default = 0)

0 Quiet
1 Summary
2 Full log

**SolveEVProb (integer):** Solve and report the expected value solution
(default = 0)

**SolveScenarios (integer):** Maximum number of scenarios solved
(default = 100)

**SubSolvePar (string):** User defined GAMS parameters for subsolve

**SubSolver (string):** Subsolver to run
Decis using Cplex
Decis using Minos

**Terminate (integer):** Generate the GAMS source code of the reformulated Model
(default = 0)

**TolBen (real):** Tolerance for stopping the decomposition algorithm
(default = 1e-7)

**TolW (real):** tolerance when writing debug solution output
(default = 1e-9)

## 4 Appendix A - GAMS/DECIS Illustrative Examples

### 4.1 Example APL1P

* APL1P test model
* Dr. Gerd Infanger, November 1997

```gams
set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter ccmin(g) min capacity / g1 1000, g2 1000 /;
parameter ccmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

table f(g,dl) operating cost
   h  m  l
  g1  4.3 2.0 0.5
  g2  8.7 4.0 1.0;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost total cost;
positive variable x(g) capacity of generators;
positive variable y(g, dl) operating level;
positive variable s(dl) unserved demand;

equations
    cost total cost
    cmin(g) minimum capacity
    cmax(g) maximum capacity
    omax(g) maximum operating level
    demand(dl) satisfy demand;

    cost .. tcost =e= sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

    cmin(g) .. x(g) =ge= ccmin(g);
    cmax(g) .. x(g) =le= ccmax(g);
    omax(g) .. sum(dl, y(g,dl)) =le= alpha(g)*x(g);
    demand(dl) .. sum(g, y(g,dl)) + s(dl) =ge= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g) = 1;
y.stage(g, dl) = 2;
s.stage(dl) = 2;
cmin.stage(g) = 1;
```

cmax.stage(g) = 1;
omax.stage(g) = 2;
demand.stage(dl) = 2;

* defining independent stochastic parameters
set stoch /out, pro/;
set omega1 / o11, o12, o13, o14/;
table v1(stoch, omega1)
  o11 o12 o13 o14
  out -1.0 -0.9 -0.5 -0.1
  pro 0.2 0.3 0.4 0.1
;
set omega2 / o21, o22, o23, o24, o25/;
table v2(stoch, omega2)
  o21 o22 o23 o24 o25
  out -1.0 -0.9 -0.7 -0.1 -0.0
  pro 0.1 0.2 0.5 0.1 0.1
;
set omega3 / o31, o32, o33, o34/;
table v3(stoch, omega1)
  o11 o12 o13 o14
  out 900 1000 1100 1200
  pro 0.15 0.45 0.25 0.15
;
set omega4 / o41, o42, o43, o44/;
table v4(stoch, omega1)
  o11 o12 o13 o14
  out 900 1000 1100 1200
  pro 0.15 0.45 0.25 0.15
;
set omega5 / o51, o52, o53, o54/;
table v5(stoch, omega1)
  o11 o12 o13 o14
  out 900 1000 1100 1200
  pro 0.15 0.45 0.25 0.15
;
* defining distributions
file stg /MODEL.STG/;
put stg;
put "INDEP DISCRETE" /;
loop(omega1,
  put "x g1 omax g1 ", v1("out", omega1), " period2 ", v1("pro", omega1) /);
); put "+" /;
loop(omega2,
  put "x g2 omax g2 ", v2("out", omega2), " period2 ", v2("pro", omega2) /);
); put "+" /;
loop(omega3,
  put "RHS demand h ", v3("out", omega3), " period2 ", v3("pro", omega3) /);
); put "+" /;
loop(omega4,
  put "RHS demand m ", v4("out", omega4), " period2 ", v4("pro", omega4) /);
); put "+" /;
loop(omega5,
  put "RHS demand l ", v5("out", omega5), " period2 ", v5("pro", omega5) /);
); putclose stg;

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX

option lp=decism;
aplip.optfile = 1;

solve aplip using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
4.2 Example APL1PCA

* APL1PCA test model
* Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter ccmin(g) min capacity / g1 1000, g2 1000 /;
parameter ccmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

table f(g,dl) operating cost
        h   m   l
    g1  4.3  2.0  0.5
    g2  8.7  4.0  1.0;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost total cost;
positive variable x(g) capacity of generators;
positive variable y(g, dl) operating level;
positive variable s(dl) unserved demand;

equations
cost total cost
cmin(g) minimum capacity
cmax(g) maximum capacity
omax(g) maximum operating level
demand(dl) satisfy demand;

cost .. tcost =e= sum(g, c(g)*x(g))
    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
    + sum(dl,us(dl)*s(dl));

cmin(g) .. x(g) =g= ccmin(g);
cmax(g) .. x(g) =l= ccmax(g);
omax(g) .. sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g) = 1;
y.stage(g, dl) = 2;
s.stage(dl) = 2;
cmin.stage(g) = 1;
cmax.stage(g) = 1;
omax.stage(g) = 2;
demand.stage(dl) = 2;

* defining independent stochastic parameters
set stoch /out, pro/;
set omega1 / o11, o12 /;
table v1(stoch,omega1)
    o11  o12
out  2.1  1.0
pro  0.5  0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
    o21  o22
out  2.0  1.0
pro  0.2  0.8 ;

parameter hm1(dl) / h 300., m 400., l 200. /;
parameter hm2(dl) / h 100., m 150., l 300. /

* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /
put stg;

put "BLOCKS DISCRETE" /
scalar h1;
loop(omega1,
  put "BL v1 period2 ", v1("pro", omega1)/
  loop(dl,
    h1 = hm1(dl) * v1("out", omega1);
    put "RHS demand ", dl.tl:1, " ", h1/
  );
);
loop(omega2,
  put " BL v2 period2 ", v2("pro", omega2) /
  loop(dl,
    h1 = hm2(dl) * v2("out", omega2);
    put "RHS demand ", dl.tl:1, " ", h1/
  );
);
putclose stg;

* setting DECIS as optimizer
DECIS uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;
solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
5 Appendix B - Error Messages

1. ERROR in MODEL.STO: kwd, word1, word2 was not matched in first realization of block
   The specification of the stochastic parameters is incorrect. The stochastic parameter has not been specified in the
   specification of the first outcome of the block. When specifying the first outcome of a block always include all
   stochastic parameters corresponding to the block.

2. Option word1 word2 not supported
   You specified an input distribution in the stochastic file that is not supported. Check the DECIS manual for supported
   distributions.

3. Error in time file
   The time file is not correct. Check the file MODEL.TIM. Check the DECIS manual for the form of the time file.

4. ERROR in MODEL.STO: stochastic RHS for objective, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the objective
   row (row name2). Check file MODEL.STO.

5. ERROR in MODEL.STO: stochastic RHS in master, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the master
   problem (row name2). Check file MODEL.STO.

6. ERROR in MODEL.STO: col not found, name1
   The specification in the stochastic file is incorrect. The entry in the stochastic file, name1, is not found in the core file.
   Check file MODEL.STO.

7. ERROR in MODEL.STO: invalid col/row combination, (name1/name2)
   The stochastic file (MODEL.STO) contains an incorrect specification.

8. ERROR in MODEL.STO: no nonzero found (in B or D matrix) for col/row (name1, name2)
   There is no nonzero entry for the combination of name1 (col) and name2 (row) in the B-matrix or in the D-matrix.
   Check the corresponding entry in the stochastic file (MODEL.STO). You may want to include a nonzero coefficient for
   (col/row) in the core file (MODEL.COR).

9. ERROR in MODEL.STO: col not found, name2
   The column name you specified in the stochastic file (MODEL.STO) does not exist in the core file (MODEL.COR).
   Check the file MODEL.STO.

10. ERROR in MODEL.STO: stochastic bound in master, col name2
    You specified a stochastic bound on first-stage variable name2. Check file MODEL.STO.

11. ERROR in MODEL.STO: invalid bound type (kwd) for col name2
    The bound type, kwd, you specified is invalid. Check file MODEL.STO.

12. ERROR in MODEL.STO: row not found, name2
    The specification in the stochastic file is incorrect. The row name, name2, does not exist in the core file. Check file
        MODEL.STO.

13. ERROR: problem infeasible
    The problem solved (master- or subproblem) turned out to be infeasible. If a subproblem is infeasible, you did not
    specify the problem as having the property of "complete recourse". Complete recourse means that whatever first-stage
    decision is passed to a subproblem, the subproblem will have a feasible solution. It is the best way to specify a
    problem, especially if you use a sampling based solution strategy. If DECIS encounters a feasible subproblem, it adds
    a feasibility cut and continues the execution. If DECIS encounters an infeasible master problem, the problem you
    specified is infeasible, and DECIS terminates. Check the problem formulation.

14. ERROR: problem unbounded
    The problem solved (master- or subproblem) turned out to be unbounded. Check the problem formulation.

15. ERROR: error code: inform
    The solver returned with an error code from solving the problem (master- or subproblem). Consult the users’ manual
    of the solver (MINOS or CPLEX) for the meaning of the error code, inform. Check the problem formulation.
16. ERROR: while reading SPECS file
   The MINOS specification file (MINOS.SPC) contains an error. Check the specification file. Consult the MINOS
   user’s manual.

17. ERROR: reading mps file, mpsfile
   The core file mpsfile (i.e., MODEL.COR) is incorrect. Consult the DECIS manual for instructions regarding the MPS
   format.

18. ERROR: row 1 of problem ip is not a free row
   The first row of the problem is not a free row (i.e., is not the objective row). In order to make the fist row a free row, set
   the row type to be ’N’. Consult the DECIS manual for the MPS specification of the problem.

19. ERROR: name not found = nam1, nam2
   There is an error in the core file (MODEL.COR). The problem cannot be decomposed correctly. Check the core file
   and check the model formulation.

20. ERROR: matrix not in staircase form
   The constraint matrix of the problem as specified in core file (MODEL.COR) is not in staircase form. The first-stage
   rows and columns and the second-stage rows and columns are mixed within each other. Check the DECIS manual as to
   how to specify the core file. Check the core file and change the order of rows and columns.
6 DECIS License and Warranty

The software, which accompanies this license (the "Software") is the property of Gerd Infanger and is protected by copyright law. While Gerd Infanger continues to own the Software, you will have certain rights to use the Software after your acceptance of this license. Except as may be modified by a license addendum, which accompanies this license, your rights and obligations with respect to the use of this Software are as follows:

- You may
  1. Use one copy of the Software on a single computer,
  2. Make one copy of the Software for archival purposes, or copy the software onto the hard disk of your computer and retain the original for archival purposes,
  3. Use the Software on a network, provided that you have a licensed copy of the Software for each computer that can access the Software over that network, item After a written notice to Gerd Infanger, transfer the Software on a permanent basis to another person or entity, provided that you retain no copies of the Software and the transferee agrees to the terms of this agreement.

- You may not
  1. Copy the documentation, which accompanies the Software,
  2. Sublicense, rent or lease any portion of the Software,
  3. Reverse engineer, de-compile, disassemble, modify, translate, make any attempt to discover the source code of the Software, or create derivative works from the Software.

6.1 Limited Warranty:

Gerd Infanger warrants that the media on which the Software is distributed will be free from defects for a period of thirty (30) days from the date of delivery of the Software to you. Your sole remedy in the event of a breach of the warranty will be that Gerd Infanger will, at his option, replace any defective media returned to Gerd Infanger within the warranty period or refund the money you paid for the Software. Gerd Infanger does not warrant that the Software will meet your requirements or that operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

6.2 Disclaimer of Damages:

REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL GERD INFANGER BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF GERD INFANGER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IN NO CASE SHALL GERD INFANGER’S LIABILITY EXCEED THE PURCHASE PRICE FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept the Software.

6.3 General:

This Agreement will be governed by the laws of the State of California. This Agreement may only be modified by a license addendum, which accompanies this license or by a written document, which has been signed by both you and Gerd Infanger. Should you have any questions concerning this Agreement, or if you desire to contact Gerd Infanger for any reason, please write:

Gerd Infanger, 1590 Escondido Way, Belmont, CA 94002, USA.
Bibliography


**DICOPT**

Ignacio E. Grossmann, Jagadisan Viswanathan, Aldo Vecchietti; Engineering Research Design Center, Carnegie Mellon University, Pittsburgh, PA

Ramesh Raman, Erwin Kalvelagen; GAMS Development Corporation, Washington D.C.

### Contents

1. **Introduction** ........................................... 189
2. **Requirements** ........................................ 190
3. **How to Run a Model with GAMS/DICOPT** .......... 190
4. **Overview of DICOPT** .................................. 190
5. **The Algorithm** ......................................... 191
6. **Modeling** ............................................... 192
   6.1 Relaxed Model ........................................ 192
   6.2 OPTCR and OPTCA .................................... 194
   6.3 Integer Formulations ................................. 194
   6.4 Non-smooth Functions ............................... 194
7. **GAMS Options** ......................................... 195
   7.1 The OPTION Statement ............................. 195
   7.2 The Model Suffix ..................................... 196
8. **DICOPT Options** ....................................... 197
   8.1 Dicopt algorithmic options ......................... 197
   8.2 Tolerances ........................................... 197
   8.3 MIP masterproblem options ......................... 198
   8.4 NLP subproblem options ............................ 198
9. **DICOPT Output** ......................................... 204
10. **Special Notes** ........................................ 206
    10.1 Stopping Rule ...................................... 206
    10.2 Solving the NLP Problems ......................... 207
    10.3 Solving the MIP Master Problems ................. 207
    10.4 Model Debugging ................................. 208
**Bibliography** ........................................... 208

### 1 Introduction

**DICOPT** is a program for solving mixed-integer nonlinear programming (MINLP) problems that involve linear binary or integer variables and linear and nonlinear continuous variables. While the modeling and solution of MINLP optimization problems has not yet reached the stage of maturity and reliability achieved by linear, integer, or non-linear programming modeling, these problems still have rich areas of application. For example, they often arise in engineering design, management sciences, and finance. **DICOPT** (DIscrete and Continuous OPTimizer) was developed by J. Viswanathan and Ignacio E.
Grossmann at the Engineering Design Research Center (EDRC) at Carnegie Mellon University. The program is based on the extensions of the outer-approximation algorithm for the equality relaxation strategy. The MINLP algorithm inside DICOPT solves a series of NLP and MIP sub-problems. These sub-problems can be solved using any NLP (Nonlinear Programming) or MIP (Mixed-Integer Programming) solver that runs under GAMS. The performance will heavily depend on the choice of the selected subsolvers.

Although the algorithm has provisions to handle non-convexities, it does not necessarily obtain the global optimum. The GAMS/DICOPT system has been designed with two main goals in mind:

- to build on existing modeling concepts, introduce a minimum of extensions to the existing modeling language, and provide upward compatibility to ensure easy transition from existing modeling applications to nonlinear mixed-integer formulations
- to use existing optimizers to solve the DICOPT sub-problems. This allows one to match the best algorithms to the problem at hand and guarantees that any new development and enhancements in the NLP and MIP solvers become automatically and immediate available to DICOPT.

2 Requirements

In order to use DICOPT you will need to have access to a licensed GAMS BASE system as well as at least one licensed MIP solver and one licensed NLP solver. For difficult models it is advisable to also have access to multiple solvers.

3 How to Run a Model with GAMS/DICOPT

DICOPT is only able to solve MINLP and MIQCP models. If you did not specify DICOPT as the default solver you can use the following statement in your GAMS model:

```plaintext
option minlp = dicopt;
```

This should appear before the solve statement. DICOPT automatically uses the default MIP and NLP solver to solve its sub-problems. One can override this with GAMS statements like:

```plaintext
option nlp = conopt;  { or any other nlp solver }
option mip = gurobi;  { or any other mip solver }
```

These options can also be specified on the command line:

```plaintext
> gams mymodel minlp=dicopt nlp=conopt mip=gurobi
```

In the Integrated Development Environment (IDE) the command line option can be specified in the edit line in the right upper corner of the main window.

Possible NLP solvers include conopt, ipopt, knitro, minos, and snopt. Possible MIP solvers include cplex, gurobi, and xpress.

With an option file it is even possible to use alternate solvers in different cycles. Section DICOPT Options explains this is in detail.

4 Overview of DICOPT

DICOPT solves models of the form:
where \( x \) are the continuous variables and \( y \) are the discrete variables. The symbol \( \sim \) is used to denote a vector of relational operators \( \{\leq, =, \geq\} \). The constraints can be either linear or non-linear. Bounds \( \ell \) and \( u \) on the variables are handled directly. \( \lceil x \rceil \) indicates the smallest integer, greater than or equal to \( x \). Similarly, \( \lfloor x \rfloor \) indicates the largest integer, less than or equal to \( x \). The discrete variables can be either integer variables or binary variables.

5 The Algorithm

The algorithm in DICOPT is based on three key ideas:

- Outer Approximation
- Equality Relaxation
- Augmented Penalty

Outer Approximation refers to the fact that the surface described by a convex function lies above the tangent hyper-plane at any interior point of the surface. (In 1-dimension, the analogous geometrical result is that the tangent to a convex function at an interior point lies below the curve). In the algorithm outer-approximations are attained by generating linearizations at each iterations and accumulating them in order to provide successively improved linear approximations of nonlinear convex functions that underestimate the objective function and overestimate the feasible region.

Equality Relaxation is based on the following result from non-linear programming. Suppose the MINLP problem is formulated in the form:

\[
\begin{align*}
\text{minimize or maximize} & \quad f(x) + c^T y \\
\text{subject to} & \quad G(x) + H y \sim b \\
& \quad \ell \leq x \leq u \\
& \quad y \in \{0, 1\}
\end{align*}
\]

i.e. the discrete variables are binary variables and they appear linearly in the model.

Let \( y^{(0)} \) be any fixed binary vector and let \( x^{(0)} \) be the solution of the corresponding NLP subproblem:

\[
\begin{align*}
\text{minimize} & \quad c^T y^{(0)} + f(x) \\
\text{subject to} & \quad Ay^{(0)} + h(x) = 0 \\
& \quad By^{(0)} + g(x) \leq 0 \\
& \quad \ell \leq x \leq u
\end{align*}
\]

Further let

\[
T^{(0)} = \text{diag}(t_{i,i}) \\
t_{i,i} = \text{sign}(\lambda_i)
\]

where \( \lambda_i \) is the Lagrange multiplier of the \( i \)-th equality constraint.

If \( f \) is pseudo-convex, \( h \) is quasi-convex, and \( g \) is quasi-convex, then \( x^0 \) is also the solution of the following NLP:

\[
\begin{align*}
\text{minimize} & \quad c^T y^{(0)} + f(x) \\
\text{subject to} & \quad T^{(0)}(Ay^{(0)} + h(x)) \leq 0 \\
& \quad By^{(0)} + g(x) \leq 0 \\
& \quad \ell \leq x \leq u
\end{align*}
\]
In colloquial terms, under certain assumptions concerning the convexity of the nonlinear functions, an equality constraint can be relaxed to be an inequality constraint. This property is used in the MIP master problem to accumulate linear approximations.

Augmented Penalty refers to the introduction of (non-negative) slack variables on the right hand sides of the just described inequality constraints and the modification of the objective function when assumptions concerning convexity do not hold.

The algorithm underlying DICOPT starts by solving the NLP in which the 0-1 conditions on the binary variables are relaxed. If the solution to this problem yields an integer solution the search stops. Otherwise, it continues with an alternating sequence of nonlinear programs (NLP) called subproblems and mixed-integer linear programs (MIP) called master problems. The NLP subproblems are solved for fixed 0-1 variables that are predicted by the MIP master problem at each (major) iteration. For convex problems the master problem also provides a lower bound on the objective function. This lower bound (in the case of minimization) increases monotonically as iterations proceed due to the accumulation of linear approximations. Note that in the case of maximization this bound is an upper bound which can be used as a stopping criterion through a DICOPT option stop 1 (see section DICOPT Options). Another stopping criterion that tends to work very well for non-convex problems (and even for convex problems) is based on the heuristic: stop as soon as the NLP subproblems start worsening (i.e. the current NLP subproblem has an optimal objective function that is worse than the previous NLP subproblem). This stopping criterion relies on the use of the augmented penalty, and is used in the description of the algorithm below. This is also the default stopping criterion in the implementation of DICOPT. The algorithm can be stated briefly as follows:

1. Solve the NLP relaxation of the MINLP program. If \( y^{(0)} = y \) is integer, stop (integer optimum found). Else continue with step 2.

2. Find an integer point \( y^{(1)} \) with an MIP master problem that features an augmented penalty function to find the minimum over the convex hull determined by the half-spaces at the solution \((x^{(0)}, y^{(0)})\).

3. Fix the binary variables \( y = y^{(1)} \) and solve the resulting NLP. Let \((x^{(1)}, y^{(1)})\) be the corresponding solution.

4. Find an integer solution \( y^{(2)} \) with a MIP master problem that corresponds to the minimization over the intersection of the convex hulls described by the half-spaces of the KKT points at \( y^{(0)} \) and \( y^{(1)} \).

5. Repeat steps 3 and 4 until there is an increase in the value of the NLP objective function. (Repeating step 4 means augmenting the set over which the minimization is performed with additional linearizations - i.e. half-spaces - at the new KKT point).

In the MIP problems integer cuts are added to the model to exclude previously determined integer vectors \( y^{(1)}, y^{(2)}, \ldots, y^{(K)} \). For a detailed description of the theory and references to earlier work, see [1–3].

The algorithm has been extended to handle general integer variables and integer variables appearing nonlinearly in the model.

## 6 Modeling

### 6.1 Relaxed Model

Before solving a model with DICOPT, the user is strongly advised to experiment with the relaxed model where the integer restrictions are ignored. This is the RMINLP model. As the DICOPT will start solving the relaxed problem and can use an existing relaxed optimal solution, it is a good idea to solve the RMINLP always before attempting to solve the MINLP model, i.e., the following fragment is not detrimental with respect to performance:

```plaintext
model m /all/;
option nlp=conopt;
option mip=cplex;
option rminlp=conopt;
option minlp=dicopt;
*
* solve relaxed model
*```
solve m using rminlp minimizing z;
abort$(m.modelstat > 2.5) "Relaxed model could not be solved";
*
* solve minlp model
*   solve m using minlp minimizing z;

The second SOLVE statement will only be executed if the first SOLVE was successful, i.e., if the model status was one (optimal) or two (locally optimal).

In general it is not a good idea to try to solve an MINLP model if the relaxed model cannot be solved reliably. As the RMINLP model is a normal NLP model, some obvious points of attention are:

- Scaling. If a model is poorly scaled an NLP solver may not be able find the optimal or even a feasible solution. Some NLP solvers have automatic scaling algorithms, but often it is better to attack this problem on the modeling level. The GAMS scaling facility can help in this respect.

- Starting point. If a poor starting point is used, the NLP solver may not be able to find a feasible or optimal solution. A starting point can be set by setting level values, e.g. X.L = 1;. The GAMS default levels are zero, with is often not a good choice.

- Adding bounds. Add bounds so that all functions can be properly evaluated. If you have a function $\sqrt{x}$ or log($x$) in the model, you may want to add a bound X.LO=0.001;. If a function like log($f(x)$) is used, you may want to introduce an auxiliary variable and equation $y = f(x)$ with an appropriate bound Y.LO=0.001;.

In some cases the relaxed problem is the most difficult model. If more than one NLP solver is available you may want to try them in a sequence:

model m /all/;
option nlp=conopt;
option mip=cplex;
option rminlp=conopt;
option minlp=dicopt;
*
* solve relaxed model
*   solve m using rminlp minimizing z;
   if (m.modelstat > 2.5,
      option rminlp=minos;
      solve m using rminlp minimizing z;
   );
   if (m.modelstat > 2.5,
      option rminlp=snopt;
      solve m using rminlp minimizing z;
   );
*
* solve minlp model
*   solve m using minlp minimizing z;

In this fragment we first try to solve the relaxed model using CONOPT. If that fails we try MINOS, and if that solve also fails we try SNOPT.

It is worthwhile to spend some time getting the relaxed model to solve reliably and speedily. In most cases, modeling improvements in the relaxed model, such as scaling, will also benefit the subsequent NLP sub-problems. In general these modeling improvements turn out to be rather solver independent: changes that improve the performance with CONOPT will also help solving the model with MINOS.
6.2 OPTCR and OPTCA

The DICOPT algorithm assumes that the integer sub-problems are solved to optimality. The GAMS options for OPTCR and OPTCA are therefore ignored: subproblems are solved with both tolerances set to zero. If you want to solve a MIP sub-problem with an optimality tolerance you can use the DICOPT option file to set OPTCR or OPTCA. For more information see section DICOPT Options.

For models with many discrete variables, it may be necessary to introduce an OPTCR or OPTCA option in order to solve the model in acceptable time. For models with a limited number of integer variables the default to solve MIP sub-models to optimality may be acceptable.

6.3 Integer Formulations

A number of MIP formulations are not very obvious and pose a demand on the modeler's knowledge and experience. A good overview of integer programming modeling is given in [4].

Many integer formulations use a so-called big-\(M\) construct. It is important to choose small values for those big-\(M\) numbers. As an example consider the fixed charge problem where \(y_i \in \{0, 1\}\) indicate if facility \(i\) is open or closed, and where \(x_i\) is the production at facility \(i\). Then the cost function can be modeled as:

\[
C_i = f_i y_i + v_i x_i \\
x_i \leq M_i y_i \\
y_i \in \{0, 1\} \\
0 \leq x_i \leq \text{cap}_i
\]

where \(f_i\) is the fixed cost and \(v_i\) the variables cost of operating a facility \(i\). In this case the chosen \(M_i\) should be large enough that \(x_i\) is not restricted if \(y_i = 1\). On the other hand, it should be as small as possible. This leads to a choice to have \(M_i\) equal to the (tight) upperbound of variable \(x_i\) (i.e. the capacity \(\text{cap}_i\) of facility \(i\)).

6.4 Non-smooth Functions

GAMS alerts NLP modelers against the use of non-smooth functions such as \(\min()\), \(\max()\), \(\text{smin}()\), \(\text{smax}()\) and \(\text{abs}()\). In order to use these functions, a non-linear program needs to be declared as a DNLP model instead of a regular NLP model:

```verbatim
option dnlp=conopt;
model m /all/;
solve m minimizing z using dnlp;
```

This construct will warn the user that problems may arise due to the use of non-smooth functions.

A possible solution is to use a smooth approximation. For instance, the function \(f(x) = |x|\) can be approximated by \(g(x) = \sqrt{x^2 + \varepsilon}\) for some \(\varepsilon > 0\). This approximation does not contain the point \((0, 0)\). An alternative approximation can be devised that has this property:

\[
f(x) \approx \frac{2x}{1 + e^{-x/\varepsilon}} - x
\]

MINLP models do not have such protection against non-smooth functions, but the use of such functions is just as problematic here. It is possible to use discrete variables with MINLP models in order to model if-then-else situations. For instance, in the case of the absolute value we can replace \(x\) by \(x^+ - x^-\) and \(|x|\) by \(x^+ + x^-\) by using:

\[
\begin{align*}
x &= x^+ - x^- \\
|x| &= x^+ + x^- \\
x^+ &\leq \delta M \\
-x^- &\leq (1 - \delta)M \\
x^+, x^- &\geq 0 \\
\delta &\in \{0, 1\}
\end{align*}
\]

where \(\delta\) is a binary variable.
7 GAMS Options

GAMS options are specified in the GAMS model source, using either the option statement or a model suffix.

7.1 The OPTION Statement

An option statement sets a global parameter. An option statement should appear before the solve statement, as in:

```gams
model m /all/
option iterlim=100;
solve m using minlp minimizing z;
```

Option statements that affect the behavior of DICOPT are listed below:

- **option domlim = n;**
  This option sets a limit on the total accumulated number of non-linear function evaluation errors that are allowed while solving the NLP subproblems or inside DICOPT itself. An example of a function evaluation error or domain error is taking the square root of a negative number. This situations can be prevented by adding proper bounds. The default is zero, i.e. no function evaluation errors are allowed.

  In case a domain error occurs, the listing file will contain an appropriate message, including the equation that is causing the problem. For instance:

  ```
  **** ERRORS(S) IN EQUATION loss(cc,sw)
  2 instance(s) of - UNDEFINED REAL POWER (RETURNED 0.0E+00)
  ```

  If such errors appear you can increase the DOMLIM limit, but often it is better to prevent the errors from occurring in the first place. In many cases this can be accomplished by adding appropriate bounds. Sometimes extra variables and equations need to be added to accomplish this. For instance, with an expression like \(\log(x - y)\), you may want to introduce a variable \(z > \epsilon\) and an equation \(z = x - y\), so that the expression can be rewritten as \(\log(z)\).

- **option iterlim = n;**
  This option sets a limit on the total accumulated (minor) iterations performed in the MIP and NLP subproblems.

- **option minlp = dicopt;**
  This option selects DICOPT to solve MINLP problems.

- **option mip = s;**
  This option sets the MIP solver to be used for the MIP master problems. Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.

- **option nlp = s;**
  This option sets the NLP solver to be used for the NLP sub-problems. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.

- **option optca = x;**
  This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.

- **option optcr = x;**
  This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.

- **option reslim = x;**
  This option sets a limit on the total accumulated time (in seconds) spent inside DICOPT and the subsolvers.
• **option sysout = on;**
  This option will print extra information to the listing file.

In the list above (and in the following) $n$ indicates an integer number. GAMS will also accept fractional values, which will be rounded. Options marked with an $x$ parameter expect a real number. Options with an $s$ parameter expect a string argument.

### 7.2 The Model Suffix

Some options are set by assigning a value to a model suffix, as in:

```gams
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

Model suffixes that affect the behavior of DICOPT are listed below:

- **m.dictfile = 1;**
  This option tells GAMS to write a dictionary file containing information about GAMS identifiers (equation and variables names). Such information is needed when the DICOPT option nlptracelevel is used, otherwise the option can be ignored.

- **m.iterlim = n;**
  Sets the total accumulated (minor) iteration limit. This option overrides the global iteration limit set by an option statement, e.g.,

  ```gams
  model m /all/;
m.iterlim = 100;
option iterlim = 1000;
solve m using minlp minimizing z;
  ```

  will cause DICOPT to use an iteration limit of 100.

- **m.optfile = 1;**
  This option instructs DICOPT to read an option file `dicopt.opt`. This file should be located in the current directory (or the project directory when using the GAMS IDE). The contents of the option file will be echoed to the listing file and to the screen (the log file):

  ```text
  --- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
  > maxcycles 10
  --- DICOPT: Starting major iteration 1
  ```

  If the option file does not exist, the algorithm will proceed using its default settings. An appropriate message will be displayed in the listing file and in the log file:

  ```text
  --- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
  --- DICOPT: File does not exist, using defaults...
  --- DICOPT: Starting major iteration 1
  ```

- **m.optfile = n;**
  If $n > 1$ then the option file that is read is called `dicopt.op n` (for $n = 2,...,9$) or `dicopt.o n` (for $n = 10,...,99$). E.g. `m.optfile=2;` will cause DICOPT to read `dicop.op2`.

- **m.prioropt = 1;**
  This option turns on the use of priorities on the discrete variables. Priorities influence the branching order chosen by the MIP solver during solution of the MIP master problems. The use of priorities can greatly impact MIP solver performance. The priorities themselves have to be specified using the `.prior` variables suffix, e.g. `x.prior(i,j) = ord(i);`. Contrary to intuition, variables with a lower value for their priority are branched on before variables with a higher priority, i.e., the most important variables should get lower priority values.
• $m.\text{reslim} = x$;
  
Sets the total accumulated time limit. This option overrides the global time limit set by an option statement.

8 DICOPT Options

This sections describes the options that can be specified in the DICOPT option file. This file is usually called dicopt.opt. The optfile model suffix must be set to tell DICOPT to read this file:

```gams
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

The option file is searched for in the current directory, or in the project directory when the IDE is used.

The option file is a standard text file, with a single option on each line. All options are case-insensitive. A line is a comment line if it starts with an asterisk, * in column one. A valid option file can look like:

```plaintext
* stop only on infeasible MIP or hitting a limit
stop 0
* use minos to solve first NLP sub problem
* and conopt for all subsequent ones
nlpsolver minos conopt
```

A convenient way to write the option file from within a GAMS model is to use the following construct:

```gams
$onecho > dicopt.opt
top 0
nlpsolver minos conopt
$offecho
```

This will make the model self-contained. Notice, however, that this overwrites an existing file dicopt.opt.

Available DICOPT options are listed below:

### 8.1 Dicopt algorithmic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue</td>
<td>How to proceed in case of NLP errors</td>
<td>2</td>
</tr>
<tr>
<td>infbnd</td>
<td>Bound to use for unbounded integer variables in integer cuts</td>
<td>10000</td>
</tr>
<tr>
<td>infesder</td>
<td>Use derivatives of infeasible nonlinear subproblems</td>
<td>0</td>
</tr>
<tr>
<td>maxcycles</td>
<td>Maximum number of cycles</td>
<td>20</td>
</tr>
<tr>
<td>relaxed</td>
<td>How to start DICOPT</td>
<td>1</td>
</tr>
<tr>
<td>solvelink</td>
<td>Solvelink for NLP and MIP subsolver</td>
<td>5</td>
</tr>
<tr>
<td>stop</td>
<td>Stopping criterion</td>
<td>2</td>
</tr>
<tr>
<td>weight</td>
<td>Penalty parameter</td>
<td>1000</td>
</tr>
</tbody>
</table>

### 8.2 Tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>epsmip</td>
<td>Tolerance on test on monotonic improvement of MIP master problem</td>
<td>1.0e-6</td>
</tr>
</tbody>
</table>
epsx | Tolerance for integer values when loading relaxed solution | 1.0e-3

### 8.3 MIP masterproblem options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mipiterlim</td>
<td>List of iteration limits</td>
<td></td>
</tr>
<tr>
<td>mipoptfile</td>
<td>List of option files for MIP solver</td>
<td></td>
</tr>
<tr>
<td>mipreslim</td>
<td>List of resource limits</td>
<td></td>
</tr>
<tr>
<td>mipsolver</td>
<td>List of MIP solvers</td>
<td></td>
</tr>
<tr>
<td>optca</td>
<td>List of OPTCA values</td>
<td></td>
</tr>
<tr>
<td>optcr</td>
<td>List of OPTCR values</td>
<td></td>
</tr>
</tbody>
</table>

### 8.4 NLP subproblem options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>domlim</td>
<td>List of allowed number of domain errors</td>
<td></td>
</tr>
<tr>
<td>nlpiterlim</td>
<td>List of iteration limits</td>
<td></td>
</tr>
<tr>
<td>nlpoptfile</td>
<td>List of option files for NLP solver</td>
<td></td>
</tr>
<tr>
<td>nlpreslim</td>
<td>List of resource limits</td>
<td></td>
</tr>
<tr>
<td>nlpsolver</td>
<td>List of NLP solvers</td>
<td></td>
</tr>
<tr>
<td>nlptracefile</td>
<td>Base name of trace files</td>
<td>nlptrace</td>
</tr>
<tr>
<td>nlptracelevel</td>
<td>Trace level</td>
<td>0</td>
</tr>
</tbody>
</table>

**continue** *(integer)*: How to proceed in case of NLP errors

This option can be used to let DICOPT continue in cases of NLP solver failure. The preferred approach is to fix the model so that NLP subproblems solve without problems. In some cases, however, (partial) failures of an NLP solver in solving the NLP subproblems can be ignored, as DICOPT may recover later on. Adding the option `continue 0` during model debugging enables DICOPT to function in a more specific way.

*(default = 2)*

- **0** Stop on solver failure Stop on solver failure. DICOPT will terminate when an NLP subproblem can not be solved to optimality. Some NLP solvers terminate with a status other than optimal if not all of the termination criteria are met. For instance, the change in the objective function is negligible (indicating convergence) but the reduced gradients are not within the required tolerance. Such a solution may or may not be close to the (local) optimum. Using `continue 0` will prevent DICOPT from accepting such a solution.

- **1** Accept non-optimal feasible solutions NLP subproblem failures resulting in a non-optimal but feasible solutions are accepted. Sometimes an NLP solver cannot make further progress towards meeting all optimality conditions, although the current solution is feasible. Such a solution can be accepted by this option.

- **2** Ignore infeasible solutions NLP subproblem failures resulting in a non-optimal but feasible solution are accepted (as in option `continue 1`). NLP subproblem failures resulting in an infeasible solution are ignored. The corresponding configuration of discrete variables is forbidden to be used again. An integer cut to accomplish this is added to subsequent MIP master problems. Note that the relaxed NLP solution should be feasible. This setting is the default.

**domlim** *(string)*: List of allowed number of domain errors
domlim \( i_1 \ i_2 \ldots \ i_n \). Sets a limit of the number of function and derivative evaluation errors for a particular cycle. A number of \(-1\) means that the global GAMS option \texttt{domlim} is used. The last number \(i_n\) sets a domain error limit for all cycles \(n, n+1, \ldots\).

Example: \texttt{domlim 0 100 0}

The NLP solver in the second cycle is allowed to make up to 100 evaluation errors, while all other cycles must be solved without evaluation errors.

The default is to use the global GAMS \texttt{domlim} option.

epsmip (real): Tolerance on test on monotonic improvement of MIP master problem

This option can be used to relax the test on MIP objective functions. The objective function values of the MIP master problems should form a monotonic worsening curve. This is not the case if the MIP master problems are not solved to optimality. If the options \texttt{OPTCR} or \texttt{OPTCA} are set to a nonzero value, this test is bypassed. If the test fails, DICOPT will fail with a message:

The MIP solution became better after adding integer cuts. Something is wrong. Please check if your model is properly scaled. Also check your big M formulations -- the value of M should be relatively small. This error can also occur if you used a MIP solver option file with a nonzero \texttt{OPTCR} or \texttt{OPTCA} setting. In that case you may want to increase the EPSMIP setting using a DICOPT option file.

The value of

\[
\frac{\text{PreviousObj} - \text{CurrentObj}}{1 + |\text{PreviousObj}|}
\]

is compared against epsmip. In case the test fails but you want DICOPT to continue anyway, you may want to increase the value of epsmip. The current values used in the test (previous and current MIP objective, epsmip) are printed along with the above message to provide information about how much you should increase epsmip to pass the test. Normally, you should not have to change this value.

(default = \(1.0e-6\))

epsx (real): Tolerance for integer values when loading relaxed solution

This tolerance is used to distinguish integer variables that are set to an integer value by the user, or integer variables that are fractional. See the option \texttt{relaxed}.

(default = \(1.0e-3\))

infbnd (real): Bound to use for unbounded integer variables in integer cuts

Value to use for missing bounds on discrete variables when constructing integer cuts.

(default = \(10000\))

infeasder (integer): Use derivatives of infeasible nonlinear subproblems

This option is to determine whether linearizations of infeasible NLP subproblems are added or not added to the MIP master problem.

(default = \(0\))

\(0\) No linearizations of infeasible NLP subproblems This is the default option in which no linearizations are added in the infeasible NLP subproblems. In this case a simple integer cut is added to remove from consideration the 0-1 vector that gave rise to the infeasible NLP. Since this may slow the convergence, it is recommended to reformulate the MINLP with "elastic" constraints (i.e., adding slacks to infeasible constraints and adding a penalty for them in the objective) to ensure that the NLP subproblems are mathematically feasible.

\(1\) Add linearization for infeasible NLP subproblems This will add linearizations derived from the infeasible NLP subproblem to the master problem. This option is recommended to speed up convergence when the MINLP is known to be convex (i.e. its continuous relaxation is convex). The possibility of cutting-off the global optimum is increased if it is used for a nonconvex MINLP.
maxcycles (integer): Maximum number of cycles

The maximum number of cycles or major iterations performed by DICOPT.

(default = 20)

mipiterlim (string): List of iteration limits

mipiterlim $i_1$ $i_2$ ... $i_n$ sets an iteration limit on individual MIP master problems. The last number $i_n$ is valid for all subsequent cycles $n, n+1, \ldots$. A number of $-1$ indicates that there is no (individual) limit on the corresponding MIP master problem. A global iteration limit is maintained through the GAMS option iterlim.

Example: mipiterlim 10000 -1
The first MIP master problem cannot use more than 10000 iterations, while subsequent MIP master problems are not individually restricted.

Example: mipiterlim 10000
Sets an iteration limit of 10000 on all MIP master problems.

When this option is used it is advised to have the option continue set to its default of 2. The default for this option is not to restrict iteration counts on individual solves of MIP master problems. The default for this option is not to restrict iteration counts on individual solves of MIP master problems.

mipoptfile (string): List of option files for MIP solver

mipoptfile $s_1$ $s_2$ ... $s_n$ specifies the option file to be used for the MIP master problems. Several option files can be specified, separated by a blank. If a digit 1 is entered the default option file for the MIP solver in question is being used. The digit 0 indicates that no option file is to be used. The last option file is also used for subsequent MIP master problems.

Example: mipoptfile mip.opt mip2.opt 0
This option will cause the first MIP master problem solver to read the option file mip.opt and the second one to read the option file mip2.opt; subsequent MIP master problem solvers will not use any option file.

Example: mipoptfile 1
This will cause the MIP solver for all MIP subproblems to read a default option file (e.g. cplex.opt, xpress.opt, gurobi.opt etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

mipreslim (string): List of resource limits

mipreslim $x_1$ $x_2$ ... $x_n$ sets a resource (time) limit on individual MIP master problems. The last number $x_n$ is valid for all subsequent cycles $n, n+1, \ldots$. A number $-1.0$ means that the corresponding MIP master problem is not individually time restricted. A global time limit is maintained through the GAMS option reslim.

Example: mipreslim -1 10000 -1
The MIP master problem in cycle 2 cannot use more than 100 seconds, while subsequent MIP master problems are not individually restricted.

Example: mipreslim 1000
Sets a time limit on all MIP master problems of 1000 seconds.

When this option is used it is advised to have the option continue set to its default of 2. The default for this option is not to restrict individually the time a solver can spent on the MIP master problem.

mipsolver (string): List of MIP solvers

This option specifies with MIP solver to use for the MIP master problems.

Example: mipsolver cplex xpress
This instructs DICOPT to use Cplex for the first MIP and XPRESS for the second and subsequent MIP problems. The last entry may be used for more than one problem.

The names to be used for the solvers are the same as one uses in the GAMS statement OPTION MIP=....; The default is to use the default MIP solver.
Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.

**nlpiterlim (string):** List of iteration limits

*nlpiterlim* $i_1$ $i_2$ ... $i_n$ sets an iteration limit on individual NLP subproblems. The last number $i_n$ is valid for all subsequent cycles $n, n+1, ...$. A number of $-1$ indicates that there is no (individual) limit on the corresponding NLP subproblem. A global iteration limit is maintained through the GAMS option *reslim*.

Example: *nlpiterlim* 1000 -1
The first (relaxed) NLP subproblem cannot use more than 1000 iterations, while subsequent NLP subproblems are not individually restricted.

Example: *nlpiterlim* 1000
Sets an iteration limit of 1000 on all NLP subproblems.

When this option is used it is advised to have the option continue set to its default of 2. This default does not restrict the amount of iterations an NLP solver can spend on an NLP subproblem, other than the global iteration limit.

**nlpoptfile (string):** List of option files for NLP solver

*nlpoptfile* $s_1$ $s_2$ ... $s_n$ specifies the option file to be used for the NLP subproblems. Several option files can be specified, separated by a blank. If a digit 1 is entered, the default option file for the NLP solver in question is being used. The digit 0 indicates that no option file is to be used. The last option file is also used for subsequent NLP subproblems.

Example: *nlpoptfile* nlp.opt nlp2.opt 0
This option will cause the first NLP subproblem solver to read the option file nlp.opt and the second one to read the option file nlp2.opt; subsequent NLP subproblem solvers will not use any option file.

Example: *nlpoptfile* 1
This will cause the NLP solver for all NLP subproblems to read a default option file (e.g. conopt.opt, minos.opt, snopt.opt etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

**nlpreslim (string):** List of resource limits

*nlpreslim* $x_1$ $x_2$ ... $x_n$ sets a resource (time) limit on individual NLP subproblems. The last number $x_n$ is valid for all subsequent cycles $n, n+1, ...$. A number $-1.0$ means that the corresponding NLP subproblem is not individually time restricted. A global time limit is maintained through the GAMS option *reslim*.

Example: *nlpreslim* 100 -1
The first (relaxed) NLP subproblem cannot use more than 100 seconds, while subsequent NLP subproblems are not individually restricted.

Example: *nlpreslim* 1000
Sets a time limit of 1000 seconds on all NLP subproblems.

When this option is used it is advised to have the option *continue* set to its default of 2. This default does not restrict the time an NLP solver can spend on an NLP subproblem (other than the global resource limit).

**nlpsolver (string):** List of NLP solvers

*nlpsolver* $s_1$ $s_2$ ... $s_n$. This option specifies which NLP solver to use for the NLP subproblems.

Example: *nlpsolver* conopt minos snopt
tells DICOPT to use CONOPT for the relaxed NLP, MINOS for the second NLP subproblem, and SNOPT for the third and subsequent ones. The last entry is used for more than one subproblem: for all subsequent ones DICOPT will use the last specified solver.

The names to be used for the solvers are the same as those used in the GAMS statement*OPTION NLP=...*.
The default is to use the default NLP solver. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.
**nlptracefile** *(string)*: Base name of trace files

Name of the files written if the option `nlptracelevel` is set. Only the stem is needed: if the name is specified as `nlptracefile nlptrace`, then files of the form `nlptrace.001`, `nlptrace.002`, etc. are written. These files contain the settings of the integer variables so that NLP subproblems can be investigated independently of DICOPT.

*(default = nlptrace)*

**nlptracelevel** *(integer)*: Trace level

This sets the level for NLP tracing, which writes a file for each NLP sub-problem, so that NLP sub-problems can be investigated outside the DICOPT environment. See also the option DICOPTnlptracefile “nlptracefile”.

By including a trace file in your original problem and changing it into an MINLP problem, the subproblem will be solved directly by an NLP solver. This option only works if the names in the model (names of variables and equations) are exported by GAMS. This can be accomplished by using the `m.dictfile` model suffix, as in `m.dictfile=1;`. In general it is more convenient to use the CONVERT solver to generate isolated NLP models (see section Model Debugging).

*(default = 0)*

- **0** No trace info is written. No trace files are written. This is the default.
- **1** GAMS file with fixed integer variables. A GAMS file for each NLP subproblem is written which fixes the discrete variables.
- **2** Include levels of continuous variables. As `nlptracelevel 1`, but in addition level values of the continuous variables are written.
- **3** Include all levels and marginals. As `nlptracelevel 2`, but in addition marginal values for the equations and variables are written.

**optca** *(string)*: List of OPTCA values

`optca x_1 x_2 ... x_n`. The absolute optimality criterion for the MIP master problems. The GAMS option `optca` is ignored, as, by default, DICOPT wants to solve MIP master problems to optimality. It is possible to stop the MIP solver earlier to allow it to solve a large problem, by specifying a value for `optca` or `optcr` in a DICOPT option file. With setting a value for `optca`, the MIP solver is instructed to stop as soon as the gap between the best possible integer solution and the best found integer solution is less than `x`, i.e. stop as soon as

\[ |\text{BestFound} - \text{BestPossible}| \leq x \]

It is possible to specify a different `optca` value for each cycle. The last number `x_n` is valid for all subsequent cycles `n, n + 1, ...`

Example: `optca 10`
Stop the search in all MIP problems as soon as the absolute gap is less than 10.

Example: `optca 0 10 0`
Sets a nonzero `optca` value of 10 for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

**optcr** *(string)*: List of OPTCR values

`optcr x_1 x_2 ... x_n`. The relative optimality criterion for the MIP master problems. The GAMS option `optcr` is ignored, as by default DICOPT wants to solve MIP master problems to optimality. To allow it to solve a large problem it is possible to stop the MIP solver earlier by specifying a value for `optca` or `optcr` in a DICOPT option file. With setting a value for `optcr`, the MIP solver is instructed to stop as soon as the relative gap between the best possible integer solution and the best found integer solution is less than `x`, i.e., stop as soon as

\[ \frac{|\text{BestFound} - \text{BestPossible}|}{|\text{BestPossible}|} \leq x \]

Note that the relative gap cannot be evaluated if the best possible integer solution is zero. In these cases the
absolute optimality criterion optca can be used. It is possible to specify a different optcr value for each cycle. The last number $x_n$ is valid for all subsequent cycles $n, n+1, \ldots$.

Example: optcr 0.1
Stop the search in all the MIP problems as soon as the relative gap is smaller than 10%.

Example: optcr 0 0.01 0
Sets a nonzero optcr value of 1% for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

**relaxed (integer):** How to start DICOPT

In some cases it may be possible to use a known configuration of the discrete variables. Some users have very difficult problems, where the relaxed problem cannot be solved but where NLP sub-problems with the integer variables fixed are much easier. If a reasonable integer configuration is known in advance in theses cases we can bypass the relaxed NLP and tell DICOPT to directly start with this integer configuration. The integer variables need to be specified by the user before the solve statement by assigning values to the levels, as in $Y.L(I) = \text{INITVAL}(I)$.

(default = 1)

0 Start with all integers fixed to the starting value The first NLP sub-problem will be executed with all integer variables fixed to the values specified by the user. If you don’t assign a value to an integer variable, it will retain it’s current value, which is zero by default.

1 Start with relaxed NLP The first NLP problem is the relaxed NLP problem: all integer variables are relaxed between their bounds. This is the default.

2 Start with mixture of fixed and relaxed integers The first NLP subproblem will be executed with some variables fixed and some relaxed. The program distinguishes the fixed from the relaxed variables by comparing the initial values against the bounds and the tolerance allowed $\text{EPSX}$. $\text{EPSX}$ has a default value of 1.e-3. This can be changed through the option file.

**solvelink (integer):** Solvelink for NLP and MIP subsolver

This option defines the solvelink used for the NLP and MIP subsolver.

(direct = 5)

1 Call NLP and MIP solver via script
2 Call NLP and MIP solver via module
5 Call NLP and MIP solver in memory

**stop (integer):** Stopping criterion

This option defines the stopping criterion to be used. The search is always stopped when the (minor) iteration limit (the iterlim option), the resource limit (the reslim option), or the major iteration limit (see maxcycles) is hit or when the MIP master problem becomes infeasible.

Note: In general a higher number stops earlier, although in some cases stopping rule 2 may terminate the search earlier than rule 1. Section **Modeling** shows some experiments with these stopping criteria.

(direct = 2)

0 Stop on maxcycles. Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will often find the optimal solution using one of the more optimistic stopping rules. Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will often find the optimal solution using one of the more optimistic stopping rules.
1. Stop on crossover. Stop as soon as the bound defined by the objective of the last MIP master problem is worse than the best NLP solution found (a 'crossover' occurred). For convex problems this gives a global solution, provided the weights are large enough. This stopping criterion should only be used if it is known or it is very likely that the nonlinear functions are convex. In the case of non-convex problems the bounds of the MIP master problem are not rigorous. Therefore, the global optimum can be cut off with the setting stop 1.

2. Stop on worsening. Stop as soon as the NLP subproblems stop improving. This 'worsening' criterion is a heuristic. For non-convex problems in which valid bounds can not be obtained the heuristic often works very well. Even on convex problems, in many cases it terminates the search very early while providing an optimal or a very good integer solution. The criterion is not checked before major iteration three.

3. Stop on crossover or worsening. Stop as soon as a crossover occurs or when the NLP subproblems start to worsen. (This is a combination of 1 and 2).

**weight (real):** Penalty parameter

The value of the penalty coefficients.

(default = 1000)

## 9 DICOPT Output

DICOPT generates lots of output on the screen. DICOPT itself and also the NLP and MIP solvers that handle the sub-problems write messages to the screen. The most important part is the last part of the screen output.

In this section we will discuss the output DICOPT writes to the screen and the listing file using the model procset.gms (this model is part of the GAMS model library). A DICOPT log is written and the reason why DICOPT terminated is explained.

--- DICOPT: Checking convergence
--- DICOPT: Search stopped on worsening of NLP subproblems
--- DICOPT: Log File:

<table>
<thead>
<tr>
<th>Major Objective</th>
<th>CPU time</th>
<th>Iterations</th>
<th>Evaluation Errors</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP 1</td>
<td>5.35021</td>
<td>0.05</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>MIP 1</td>
<td>2.48869</td>
<td>0.28</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>NLP 2</td>
<td>1.72097&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>MIP 2</td>
<td>2.17864</td>
<td>0.22</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>NLP 3</td>
<td>1.92310&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>MIP 3</td>
<td>1.42129</td>
<td>0.22</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>NLP 4</td>
<td>1.41100</td>
<td>0.00</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

--- DICOPT: Terminating...
--- DICOPT: Stopped on NLP worsening

The search was stopped because the objective function of the NLP subproblems started to deteriorate.

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- PROCSET.GMS(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion

Notice that the integer solutions are provided by the NLP’s except for major iteration one (the first NLP is the relaxed NLP). For all NLP’s except the relaxed one the binary variables are fixed, according to a pattern determined by the previous MIP which operates on a linearized model. The integer solutions marked with a ‘<’ are an improvement. We see that the NLP in cycle 4 starts to deteriorate, and DICOPT stops based on its default stopping rule.
Note that if the criterion stop 1 had been used the search would have been terminated at iteration 3. The reason is that the upper bound to the profit predicted by the MIP (1.42129) exceeds the best current NLP solution (1.9231). Since it can be shown that the MINLP involves convex nonlinear functions, 1.9231 is the global optimum and the criterion stop 1 is rigorous.

A similar output can be found in the listing file:

```
SOLVE SUMMARY

MODEL process OBJECTIVE pr
TYPE MINLP DIRECTION MAXIMIZE
SOLVER DICOPT FROM LINE 98

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 8 INTEGER SOLUTION
**** OBJECTIVE VALUE 1.9231

RESOURCE USAGE, LIMIT 0.771 1000.000
ITERATION COUNT, LIMIT 51 10000
EVALUATION ERRORS 0 0

--- DICOPT: Stopped on NLP worsening

The search was stopped because the objective function of the NLP subproblems started to deteriorate.
```

In case the DICOPT run was not successful, or if one of the subproblems could not be solved, the listing file will contain all
the status information provided by the solvers of the subproblems. For each iteration the configuration of the binary variables will also be printed. This extra information can also be requested via the GAMS option:

```
option sysout = on ;
```

## 10 Special Notes

This section covers some special topics of interest to users of DICOPT.

### 10.1 Stopping Rule

Although the default stopping rule behaves quite well in practice there some cases where it terminates too early. In this section we discuss the use of the stopping criteria.

When we run the example `procsel.gms` with stopping criterion 0, we see the following DICOPT log:

--- DICOPT: Starting major iteration 10  
--- DICOPT: Search terminated: infeasible MIP master problem  
--- DICOPT: Log File:

<table>
<thead>
<tr>
<th>Major Step</th>
<th>Major Iter</th>
<th>Objective Function</th>
<th>CPU time (Sec)</th>
<th>Iterations</th>
<th>Evaluation Errors</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP 1</td>
<td>1</td>
<td>5.35021</td>
<td>0.06</td>
<td>8</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 1</td>
<td>1</td>
<td>2.48869</td>
<td>0.16</td>
<td>7</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 2</td>
<td>2</td>
<td>1.72097&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 2</td>
<td>2</td>
<td>2.17864</td>
<td>0.10</td>
<td>10</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 3</td>
<td>3</td>
<td>1.92310&lt;</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 3</td>
<td>3</td>
<td>1.42129</td>
<td>0.11</td>
<td>12</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 4</td>
<td>4</td>
<td>1.41100</td>
<td>0.00</td>
<td>8</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 4</td>
<td>4</td>
<td>0.00000</td>
<td>0.22</td>
<td>23</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 5</td>
<td>5</td>
<td>0.00000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 5</td>
<td>5</td>
<td>-0.27778</td>
<td>0.16</td>
<td>22</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 6</td>
<td>6</td>
<td>-0.27778</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 6</td>
<td>6</td>
<td>-1.00000</td>
<td>0.16</td>
<td>21</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 7</td>
<td>7</td>
<td>-1.00000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 7</td>
<td>7</td>
<td>-1.50000</td>
<td>0.22</td>
<td>16</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 8</td>
<td>8</td>
<td>-1.50000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 8</td>
<td>8</td>
<td>-2.50000</td>
<td>0.11</td>
<td>16</td>
<td>0</td>
<td>cplex</td>
</tr>
<tr>
<td>NLP 9</td>
<td>9</td>
<td>-2.50000</td>
<td>0.00</td>
<td>3</td>
<td>0</td>
<td>conopt</td>
</tr>
<tr>
<td>MIP 9</td>
<td>9</td>
<td><em>Infeas</em></td>
<td>0.11</td>
<td>0</td>
<td>0</td>
<td>cplex</td>
</tr>
</tbody>
</table>

--- DICOPT: Terminating...  
--- DICOPT: Stopped on infeasible MIP

The search was stopped because the last MIP problem was infeasible. DICOPT will not be able to find a better integer solution.

--- DICOPT: Best integer solution found: 1.923099  
--- Restarting execution  
--- PROCSEL.GMS(98) 0 Mb  
--- Reading solution for model process  
*** Status: Normal completion

This example shows some behavioral features that are not uncommon for other MINLP models. First, DICOPT often finds the best integer solution in the first few major iterations. Second, in many cases as soon as the NLP’s start to give worse
integer solution no better integer solution will be found. This observation is the motivation to make stopping option 2, where DICOPT stops as soon as the NLP’s start to deteriorate, the default stopping rule. In this example DICOPT would have stopped in major iteration 4 (you can verify this in the previous section). In many cases this will give the best integer solution. For this problem, DICOPT has indeed found the global optimum.

Based on experience with other models, we find that the default stopping rule (stop when the NLP becomes worse) performs well in practice. In many cases it finds the global optimum solution for both convex and non-convex problems. In some cases, however, it may provide a sub-optimal solution. In those cases where you want more reassurance that no good integer solutions are missed you can use one of the other stopping rules.

Changing the MIP or NLP solver can change the path that DICOPT follows, since the sub-problems may have non-unique solutions. The optimum stopping rule for a particular problem depends on the MIP and NLP solvers used.

The bounds of the MIP master problem are not rigorous in the case of non-convex problems. Therefore, the global optimum can be cut-off with stop 1. However, this option is the best stopping criterion for convex problems.

10.2 Solving the NLP Problems

Using a combination of NLP solvers has been found effective in cases where the relaxed NLP and/or the other NLP sub-problems are very difficult. For example, MINOS has many more difficulties to establish if a model is infeasible, so one would like to use CONOPT for NLP subproblems that are either infeasible or barely feasible. The nlp solver option can be used to specify the NLP solver to be used for each iteration.

Infeasible NLP sub-problems can be problematic for DICOPT. Those subproblems cannot be used to form a new linearization. Effectively only the current integer configuration is excluded from further consideration by adding appropriate integer cuts, but otherwise an infeasible NLP sub-problem provides no useful information to be used by the DICOPT algorithm. If your model shows many infeasible NLP sub-problems you can try to use the infeasder option. Otherwise a strategy that can help is to introduce explicit slack variables and add them with a penalty to the objective function.

Assume your model is of the form:

\[
\begin{align*}
\min & \quad f(x,y) \\
g(x,y) & \sim b \\
\ell \leq x \leq u \\
y & \in \{0,1\}
\end{align*}
\]

where \(\sim\) is a vector of relational operators \(\{\leq, =, \geq\}\). \(x\) are continuous variables and \(y\) are the binary variables. If many of the NLP subproblems are infeasible, we can try the following elastic formulation:

\[
\begin{align*}
\min & \quad f(x,y) + M \sum_i (s_i^+ + s_i^-) \\
y & = y^B + s^+ - s^- \\
g(x,y) & \sim b \\
\ell \leq x \leq u \\
0 & \leq y \leq 1 \\
0 & \leq s^+, s^- \leq 1 \\
y^B & \in \{0,1\}
\end{align*}
\]

I.e., the variables \$y\$ are relaxed to be continuous with bounds \([0,1]\), and binary variables \$y^B\$ are introduced that are related to the variables \$y\$ through a set of the slack variables \$s^+, s^-\$. The slack variables are added to the objective with a penalty parameter \(M\). The choice of a value for \(M\) depends on the size of \(f(x,y)\), on the behavior of the model, etc. Typical values are 100 or 1000.

10.3 Solving the MIP Master Problems

MIP master problems may become expensive to solve when there are many discrete variables. One of the first things to try is to see if a different MIP solver can solve your particular problems more efficiently.
Different formulations can have dramatic impact on the performance of MIP solvers. Therefore it is advised to try out several alternative formulations. The use of priorities can have a big impact on some models. It is possible to specify a nonzero value for $OPTCA$ and $OPTCR$ in order to prevent the MIP solver from spending an unacceptable long time proving optimality of MIP master problems.

If the MIP master problem is infeasible the DICOPT solver will terminate. In this case you may want to try the same reformulation discussed in the previous paragraph.

### 10.4 Model Debugging

In this paragraph we discuss a few techniques that can be helpful in debugging your MINLP model.

- Start with solving the model as an RMINLP model. Make sure this model solves reliably before solving it as a proper MINLP model. If you have access to different NLP solvers, make sure the RMINLP model solves smoothly with all NLP solvers. CONOPT, especially, can generate useful diagnostics such as Jacobian elements (i.e. matrix elements) that become too large.

- Try different NLP and MIP solvers on the subproblems. Example: use the GAMS statement `OPTION NLP=KNITRO;` to solve all NLP subproblem using the solver KNITRO.

- The GAMS option statement `OPTION SYSOUT = ON;` can generate extra solver information that can be helpful for diagnosing problems.

- If many of the NLP subproblems are infeasible, add slacks as described in section Solving the NLP Problems.

- Run DICOPT in pedantic mode by using the DICOPT option: `CONTINUE 0`. Make sure all NLP subproblems solve to optimality.

- Don’t allow any nonlinear function evaluation errors, i.e. keep the DOMLIM limit at zero. See the discussion on DOMLIM in section The OPTION Statement.

- If you have access to another MINLP solver such as AlphaECP, Bonmin, or SBB or even global solvers like Antigone or BARON, try to use a different solver on your model. To select another solver (here SBB) use the following GAMS option statement: `OPTION MINLP=SBB;`.

- Individual NLP or MIP subproblems can be extracted from the MINLP by using the CONVERT solver, which will write a model in scalar GAMS notation that can then be solved using any GAMS NLP or MIP solver. E.g., to generate the second NLP subproblem, you can use the following DICOPT option: `NLPSOLVER CONOPT CONVERT`. The model will be written to the file `GAMS.GMS`. A disadvantage of this technique is that some precision is lost due to the fact that files are being written in plain ASCII. The advantage is that you can visually inspect these files and look for possible problems such as poor scaling.

### Bibliography


1 Introduction

The Global Mixed-Integer Quadratic Optimizer, GloMIQO (Gló-me-ko), considers Mixed-Integer Quadratically-Constrained Quadratic Programs MIQCP of the form [44, 45, 48]:

\[
\min \quad x^T \cdot Q_0 \cdot x + a_0 \cdot x \\
\text{s.t.} \quad b_m^{LO} \leq x^T \cdot Q_m \cdot x + a_m \cdot x \leq b_m^{UP} \quad \forall \ m \in \{1, \ldots, M\} \\
x \in \mathbb{R}^C \times \{0, 1\}^B \times \mathbb{Z}^I
\]

(MIQCP)

where \( C, B, I, \) and \( M \) represent the number of continuous variables, binary variables, integer variables, and constraints, respectively. Note that this model can address quadratic continuous and/or integer terms, as well as bilinear terms of continuous-continuous, integer-continuous, and integer-integer type. We assume that it is possible to infer finite bounds \([x_L^m, x_U^m] \) on the variables participating in nonlinear terms.

Major applications of MIQCP include quality blending in process networks, separating objects in computational geometry, and portfolio optimization in finance. Specific instantiations of MIQCP in process networks optimization problems include: pooling problems [1, 4, 8, 13, 20, 27–29, 36, 41–43, 46, 47, 51, 58, 59], distillation sequences [2, 21, 24], wastewater treatment and total water systems [3, 5, 10, 14, 19, 26, 30, 32, 52, 53], hybrid energy systems [11, 12, 18], heat exchanger networks [15, 23], reactor-separator-recycle systems [33, 34], separation systems [57], data reconciliation [56], batch processes [39], and crude oil scheduling [35, 37, 38, 49, 50]. Computational geometry problems formulated as MIQCP include: point packing [6, 16], cutting convex shapes from rectangles [31, 54], maximizing the area of a convex polygon [7, 9], and chip layout and compaction [17]. Portfolio optimization in financial engineering can also be formulated as MIQCP [40, 55].
Figure 9.1: Given a MIQCP optimization problem, GloMIQO reformulates the model, detects special structure in the reformulated MIQCP, solves the optimization problem, and returns the model with respect to the original problem variables.

As illustrated in Figure 9.1, GloMIQO responds dynamically to elucidate and exploit special structure within MIQCP. GloMIQO falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex MIQCP that rigorously bound the global solution, finds feasible solutions via local optimization, and divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [22, 25].

1.1 Licensing and software requirements

Using GAMS/GloMIQO requires

1. a GloMIQO or ANTIGONE license,
2. a CPLEX license, and
3. a CONOPT or SNOPT license.

GloMIQO may stand alone as a MIQCP solver; it is also available as a proper subset of the general MINLP solver ANTIGONE and is included with a GAMS/ANTIGONE license.

1.2 Running GAMS/GloMIQO

GAMS/GloMIQO solves MIQCP, RMIQCP, and QCP models. If GAMS/GloMIQO is not the default solver for these models, it can be called using the following command before the solve statement:

```
option miqcp=glomiqo, rmiqcp=glomiqo, qcp=glomiqo;
```

2 GloMIQO Options

The GloMIQO options match the GAMS/ANTIGONE options.

3 GloMIQO Algorithmic Features

As illustrated in Figure 9.1, the primary algorithmic features in GloMIQO are reformulating model input, elucidating special structure, and branch-and-bound global optimization [44, 45, 48].
Figure 9.2: (a) Process networks problems are typically defined as a series of modular units. (b) The GloMIQO variable elimination steps transform the user model. (c) The subsequent bilinear term disaggregation further reformulates the model. The entire process is seamless and unseen by the modeler; GloMIQO reverses all transformations after solving the problem and reports results with respect to the original model in (a).

Figure 9.3: (a) Nonlinear equation \( m \) is an undirected graph with nodes representing variables and edges representing nonzero coefficients \( Q_{m,i,j} \). (b) The equation is divided into separable multivariable terms by detecting disjoint vertex sets. (c) Separable multivariable terms are sum decomposable, so all high-order cuts and every bounding strategy operates on a specific multivariable term. For example, detecting three-dimensional edge-concave aggregations is illustrated in red.

### 3.1 Reformulating Model Input

While the transformation steps illustrated in Figure 9.2, are implemented generically and applied universally, the reformulations are specifically targeted at enhancing the performance of GloMIQO on process networks problems. GloMIQO effectively transforms modular process networks problems into generalized pooling problems [42, 45]. GloMIQO may also add nonconvex bilinear terms to the model formulation to generate tight Reformulation-Linearization Technique cuts.

### 3.2 Elucidating Special Structure

GloMIQO automatically detects: (a) Reformulation-Linearization Technique (RLT) equations that do not add nonlinear terms to MIQCP and (b) special structure in separable multivariable terms [45].

GloMIQO considers equation/variable and equation/equation products for generating cuts and improving variable bounding. These RLT equations are updated at every node of the branch-and-bound tree:

**Equation/Variable:** Products of variable \( x_i \) with linear equation \( m \) (e.g., \( [a_m \cdot x - b^\text{UP}_m] \cdot [x_i - x_i^\text{LO}] \leq 0 \))

**Equation/Equation:** Products of two linear equations \( m, n \) (e.g., \( -1 \cdot [a_m \cdot x - b^\text{UP}_m] \cdot [a_n \cdot x - b^\text{UP}_n] \leq 0 \))

The GloMIQO preprocessor will add particularly strong RLT cuts outright the model formulation. Modelers will significantly improve the performance of GloMIQO by writing linear constraints that can be multiplied together without increasing the number of nonlinear terms.

As depicted in Figure 9.3, GloMIQO generates an undirected graph representation of each individual nonlinear equation \( m \), partitions the equation into separable multivariable terms, and detects special structure including convexity and edge-concavity in the individual multivariable terms [45].
3.3 Branch-and-Bound Global Optimization

GloMIQO falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex MIQCP that rigorously guarantee lower bounds on the global solution, finds feasible solutions via local optimization to bound the global solution from above, and divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [22, 25].

GloMIQO generates convex relaxations using: termwise McCormick envelopes, low-dimensional edge-concave relaxations, eigenvector projections, piecewise-linear underestimators, outer approximation cuts for convex terms, and an adaptive implementation of the Reformulation-Linearization Technique (RLT) [27, 43–48].

GloMIQO dynamically tightens convex relaxations with cutting planes derived from edge-concave aggregations, αBB underestimators, and convex terms. Cuts are based on both individual equations and the collection of bilinear terms in MIQCP. The branch-and-cut strategies differentiate globally-valid αBB and convex cuts from locally-valid edge-concave cuts. Previously-generated cuts are saved in a pool and applied as appropriate in the branch-and-bound tree.

GloMIQO searches for feasible solutions by multistarting an NLP solver.

GloMIQO reduces the search space using reliability branching, feasibility-based bounds tightening, optimality-based bounds tightening, RLT-based bounds tightening, and bounds tightening based on all higher-order cuts [44, 45, 48].

Bibliography


# Gurobi 6.0

The Gurobi suite of optimization products include state-of-the-art simplex and parallel barrier solvers for linear programming (LP) and quadratic programming (QP), parallel barrier solver for quadratically constrained programming (QCP), as well as parallel mixed-integer linear programming (MILP), mixed-integer quadratic programming (MIQP) and mixed-integer quadratically constrained programming (MIQCP) solvers.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

While numerous solving options are available, Gurobi automatically calculates and sets most options at the best values for specific problems. All Gurobi options available through GAMS/Gurobi are summarized at the end of this chapter.

Next to the full GAMS/GUROBI license, we offer a GAMS/GUROBI link at a reduced price. This is the same program as the GAMS/GUROBI solver, but it uses your existing Gurobi callable library license. It is noted that the link does not work on the GUROBI stand alone license. Finally, a bare-bone interface to the LP and MIP solver of Gurobi is available under the name OSIGUROBI. It comes free of charge with any GAMS system.

## 1 Introduction

The Gurobi suite of optimization products include state-of-the-art simplex and parallel barrier solvers for linear programming (LP) and quadratic programming (QP), parallel barrier solver for quadratically constrained programming (QCP), as well as parallel mixed-integer linear programming (MILP), mixed-integer quadratic programming (MIQP) and mixed-integer quadratically constrained programming (MIQCP) solvers.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

While numerous solving options are available, Gurobi automatically calculates and sets most options at the best values for specific problems. All Gurobi options available through GAMS/Gurobi are summarized at the end of this chapter.

Next to the full GAMS/GUROBI license, we offer a GAMS/GUROBI link at a reduced price. This is the same program as the GAMS/GUROBI solver, but it uses your existing Gurobi callable library license. It is noted that the link does not work on the GUROBI stand alone license. Finally, a bare-bone interface to the LP and MIP solver of Gurobi is available under the name OSIGUROBI. It comes free of charge with any GAMS system.
2 How to Run a Model with Gurobi

The following statement can be used inside your GAMS program to specify using Gurobi

```
Option LP = Gurobi; { or MIP or RMIP or QCP or MIQCP or RMIQCP }
```

The above statement should appear before the `solve` statement. If Gurobi was specified as the default solver during GAMS installation, the above statement is not necessary.

3 Overview of GAMS/Gurobi

3.1 Linear, Quadratic and Quadratic Constrained Programming

Gurobi can solve LP and QP problems using several alternative algorithms, while the only choice for solving QCP is the parallel barrier algorithm. The majority of LP problems solve best using Gurobi’s state-of-the-art dual simplex algorithm, while most QP problems solve best using the parallel barrier algorithm. Certain types of LP problems benefit from using the parallel barrier or the primal simplex algorithms, while for some types of QP, the dual or primal simplex algorithm can be a better choice. If you are solving LP problems on a multi-core system, you should also consider using the concurrent optimizer. It runs different optimization algorithms on different cores, and returns when the first one finishes.

GAMS/Gurobi also provides access to the Gurobi infeasibility finder. The infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Gurobi reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing. The infeasibility finder is activated by the option `iis`. Another option for analyzing infeasible model the `FeasOpt` option which instructs GAMS/Gurobi to find a minimal feasible relaxation of an infeasible model. See section Feasible Relaxation for details.

GAMS/Gurobi supports sensitivity analysis (post-optimality analysis) for linear programs which allows one to find out more about an optimal solution for a problem. In particular, objective ranging and constraint ranging give information about how much an objective coefficient or a right-hand-side and variable bounds can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or the bounds and right-hand side. GAMS/Gurobi reports the sensitivity information as part of the normal solution listing. Sensitivity analysis is activated by the option `sensitivity`.

The Gurobi presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Gurobi can, in order to get better diagnostic information, rerun the problem with presolve turned off. The rerun without presolve is controlled by the option `rerun`. In default mode only problems that are small (i.e. demo sized) will be rerun.

Gurobi can either presolve a model or start from an advanced basis or primal/dual solution pair. Often the solve from scratch of a presolved model outperforms a solve from an unpresolved model started from an advanced basis/solution. It is impossible to determine a priori if presolve or starting from a given advanced basis/solution without presolve will be faster. By default, GAMS/Gurobi will automatically use an advanced basis or solution from a previous `solve` statement. The GAMS `BRatio` option can be used to specify when not to use an advanced basis/solution. The GAMS/Gurobi option `usebasis` can be used to ignore or force a basis/solution passed on by GAMS (it overrides `BRatio`). In case of multiple solves in a row and slow performance of the second and subsequent solves, the user is advised to set the GAMS `BRatio` option to 1.

3.2 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear or quadratic programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with discrete variables, Gurobi uses a branch and cut algorithm which solves a series of subproblems, LP subproblems for MILP, QP subproblems for MIQP, and QCP subproblems or LP outer approximation subproblems for MIQCP. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.
GAMS/Gurobi supports Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables. You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution.

If you specify some or all values for the discrete variables together with GAMS/Gurobi option `mipstart`, Gurobi will check the validity of the values as an integer-feasible solution. If this process succeeds, the solution will be treated as an integer solution of the current problem.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

### 3.3 Feasible Relaxation

The Infeasibility Finder identifies the causes of infeasibility by means of inconsistent set of constraints (IIS). However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Gurobi is known as FeasOpt (for Feasible Optimization) and turned on by parameter `FeasOpt` in a GAMS/Gurobi option file.

With the FeasOpt option GAMS/Gurobi accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the `.feaspref` option. The input value denotes the users willingness to relax a constraint or bound. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed. It is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a GAMS/Gurobi solver option file. The syntax is:

```
(variable or equation).feaspref(value)
```

For example, suppose we have a GAMS declaration:

```gams
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the `gurobi.opt` file can be specified by:

```
feasopt 1
v.feaspref 1
v.feaspref('i1','*') 2
v.feaspref('i1','j2') 0

e.feaspref('*','j1') 0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Futhermore, we specify that all variables v(i,j) have preference of 1, except variables over set element i1, which have a preference of 2. The variable over set element i1 and j2 has preference ~0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `~variables.feaspref 1`).
The parameter **FeasOptMode** allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter **FeasOptMode** indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations). Please check description of parameter **FeasOptMode** for details. Also check example models `feasopt∗` in the GAMS Model library.

### 3.4 Parameter Tuning Tool

The Gurobi Optimizer provides a wide variety of parameters that allow you to control the operation of the optimization engines. The level of control varies from extremely coarse-grained (e.g., the **Method** parameter, which allows you to choose the algorithm used to solve continuous models) to very fine-grained (e.g., the **MarkowitzTol** parameter, which allows you to adjust the precise tolerances used during simplex basis factorization). While these parameters provide a tremendous amount of user control, the immense space of possible options can present a significant challenge when you are searching for parameter settings that improve performance on a particular model. The purpose of the Gurobi tuning tool is to automate this search.

The Gurobi tuning tool performs multiple solves on your model, choosing different parameter settings for each, in a search for settings that improve runtime. The longer you let it run, the more likely it is to find a significant improvement.

A number of tuning-related parameters allow you to control the operation of the tuning tool. The most important is probably **TuneTimeLimit**, which controls the amount of time spent searching for an improving parameter set. Other parameters include **TuneTrials** (which attempts to limit the impact of randomness on the result), **TuneResults** (which limits the number of results that are returned), and **TuneOutput** (which controls the amount of output produced by the tool).

While parameter settings can have a big performance effect for many models, they aren’t going to solve every performance issue. One reason is simply that there are many models for which even the best possible choice of parameter settings won’t produce an acceptable result. Some models are simply too large and/or difficult to solve, while others may have numerical issues that can’t be fixed with parameter changes.

Another limitation of automated tuning is that performance on a model can experience significant variations due to random effects (particularly for MIP models). This is the nature of search. The Gurobi algorithms often have to choose from among multiple, equally appealing alternatives. Seemingly innocuous changes to the model (such as changing the order of the constraint or variables), or subtle changes to the algorithm (such as modifying the random number seed) can lead to different choices. Often times, breaking a single tie in a different way can lead to an entirely different search. We’ve seen cases where subtle changes in the search produce 100X performance swings. While the tuning tool tries to limit the impact of these effects, the final result will typically still be heavily influenced by such issues.

The bottom line is that automated performance tuning is meant to give suggestions for parameters that could produce consistent, reliable improvements on your models. It is not meant to be a replacement for efficient modeling or careful performance testing.

### 3.5 Compute Server

The Gurobi Compute Server allows you to use one or more servers to offload all of your Gurobi computations.

Gurobi compute servers support queuing and load balancing. You can set a limit on the number of simultaneous jobs each compute server will run. When this limit has been reached, subsequent jobs will be queued. If you have multiple compute servers, the current job load is automatically balanced among the available servers. By default, the Gurobi job queue is serviced in a First-In, First-Out (FIFO) fashion. However, jobs can be given different priorities (**CSPriority**). Jobs with higher priorities are then selected from the queue before jobs with lower priorities.

Gurobi Compute Server licenses and software are not included in GAMS/Gurobi. You need to get in contact with Gurobi Optimization. GAMS/Gurobi merely supports the use of a Gurobi Compute Server. Relevant options are **ComputeServer** and options starting with **CS**.
3.6 Distributed Parallel Algorithms

Gurobi Optimizer implements a number of distributed algorithms that allow you to use multiple machines to solve a problem faster. Available distributed algorithms are:

- A **distributed MIP solver**, which allows you to divide the work of solving a single MIP model among multiple machines. A manager machine passes problem data to a set of worker machines in order to coordinate the overall solution process.

- A **distributed concurrent solver**, which allows you to use multiple machines to solve an LP or MIP model. Unlike the distributed MIP solver, the concurrent solver doesn’t divide the work associated with solving the problem among the machines. Instead, each machine uses a different strategy to solve the whole problem, with the hope that one strategy will be particularly effective and will finish much earlier than the others. For some problems, this concurrent approach can be more effective than attempting to divide up the work.

- **Distributed parameter tuning**, which automatically searches for parameter settings that improve performance on your optimization model. Tuning solves your model with a variety of parameter settings, measuring the performance obtained by each set, and then uses the results to identify the settings that produce the best overall performance. The distributed version of tuning performs these trials on multiple machines, which makes the overall tuning process run much faster.

These distributed parallel algorithms are designed to be almost entirely transparent to the user. The user simply modifies a few parameters, and the work of distributing the computation to multiple machines is handled behind the scenes by Gurobi.

**Specifying the Worker Pool**

Once you’ve set up a set of one or more distributed workers, you should list their names in the WorkerPool parameter. You can provide either machine names or IP addresses, and they should be comma-separated.

You can provide the worker access password through the WorkerPassword parameter. All servers in the worker pool must have the same access password.

**Requesting Distributed Algorithms**

Once you’ve set up the worker pool through the appropriate parameters, the last step to using a distributed algorithm is to set the TuneJobs, ConcurrentJobs, or DistributedMIPJobs parameter. These parameters are used to indicate how many distinct tuning, concurrent, or distributed MIP jobs should be started on the available workers.

If some of the workers in your worker pool are running at capacity when you launch a distributed algorithm, the algorithm won’t create queued jobs. Instead, it will launch as many jobs as it can (up to the requested value), and it will run with these jobs.

These distributed algorithms have been designed to be nearly indistinguishable from the single machine versions. Our hope is that, if you know how to use the single machine version, you’ll find it straightforward to use the distributed version. The distributed algorithms respect all of the usual parameters. For distributed MIP, you can adjust strategies, adjust tolerances, set limits, etc. For concurrent MIP, you can allow Gurobi to choose the settings for each machine automatically or specify a set of options. For distributed tuning, you can use the usual tuning parameters, including TuneTimeLimit, TuneTrails, and TuneOutput.

There are a few things to be aware of when using distributed algorithms, though. One relates to relative machine performance. Distributed algorithms work best if all of the workers give very similar performance. For example, if one machine in your worker pool were much slower than the others in a distributed tuning run, any parameter sets tested on the slower machine would appear to be less effective than if they were run on a faster machine. Similar considerations apply for distributed MIP and distributed concurrent. We strongly recommend that you use machines with very similar performance. Note that if your machines have similarly performing cores but different numbers of cores, we suggest that you use the Threads parameter to make sure that all machines use the same number of cores.

Logging for distributed MIP is very similar to the standard MIP logging. The main differences are in the progress section. The header for the standard MIP logging looks like this:
By contrast, the distributed MIP header looks like this:

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Utilization</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expl Unexpl</td>
<td>Active Sync Comm</td>
<td>Incumbent BestBd Gap</td>
<td>It/Node Time</td>
</tr>
</tbody>
</table>

You’ll note that columns three through five show different information. In the distributed MIP log, these columns give information about the utilization of the distributed workers, expressed as percentages. The first of these columns shows the fraction of the preceding time period (the time since the previous progress log line) that the workers spent actively processing MIP nodes. The second column shows the time the workers spent waiting for other workers to complete tasks assigned to them. The final column shows the time spent communicating data between machines. The numbers sum to 100%.

Here is an example of a distributed MIP progress log:

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Utilization</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expl Unexpl</td>
<td>Active Sync Comm</td>
<td>Incumbent BestBd Gap</td>
<td>It/Node Time</td>
</tr>
</tbody>
</table>

Ramp-up phase complete - continuing with instance 4 (best bd 10669.8)

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Utilization</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expl Unexpl</td>
<td>Active Sync Comm</td>
<td>Incumbent BestBd Gap</td>
<td>It/Node Time</td>
</tr>
</tbody>
</table>

One thing you may find in the progress section is that node counts may not increase monotonically. In this example, a solution found at node 261 is reported before a solution found at node 0. Distributed MIP tries to create a single, unified view of node numbers, but with multiple machines processing nodes independently, possibly at different rates, some inconsistencies are inevitable.

Another difference is the line that indicates that the distributed ramp-up phase is complete. At this point, the distributed strategy transitions from a concurrent approach to a distributed approach. The log line indicates which worker was the winner in the concurrent approach. Distributed MIP continues by dividing the partially explored MIP search tree from this worker among all of the workers.

Another difference in the distributed log is in the summary section. The distributed MIP log includes a breakdown of how runtime was spent:
Runtime breakdown:
- Active: 13.73s (75%)
- Sync: 1.16s (6%)
- Comm: 3.45s (19%)

This is an aggregated view of the utilization data that is displayed in the progress log lines. In this example, the workers spent 75% of runtime actively working on MIP nodes, 6% waiting to synchronize with other workers, and 19% communicating data between machines.

4 GAMS Options

The following GAMS options are used by GAMS/Gurobi:

**Option BRatio = x;**
Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Gurobi not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed.

**Option IterLim = n;**
Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS. For MIP problems, if the number of the cumulative simplex iterations exceeds the limit, Gurobi will terminate.

**Option NodLim = x;**
Maximum number of nodes to process for a MIP problem. This GAMS option is overridden by the GAMS/Gurobi option nodelimit.

**Option OptCR = x;**
Relative optimality criterion for a MIP problem. Notice that Gurobi uses a different definition than GAMS normally uses. The OptCR option asks Gurobi to stop when

\[ |BP - BF| < |BF| \times \text{OptCR} \]

where \( BF \) is the objective function value of the current best integer solution while \( BP \) is the best possible integer solution. The GAMS definition is:

\[ |BP - BF| < |BP| \times \text{OptCR} \]

**Option ResLim = x;**
Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. Gurobi measures time in wall time on all platforms. Some other GAMS solvers measure time in CPU time on some Unix systems. This GAMS option is overridden by the GAMS/Gurobi option timelimit.

**Option SysOut = On;**
Will echo Gurobi messages to the GAMS listing file. This option may be useful in case of a solver failure.

**ModelName.Cutoff = x;**
Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than \( x \) are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm. This GAMS option is overridden by the GAMS/Gurobi option cutoff.

**ModelName.OptFile = 1;**
Instructs GAMS/Gurobi to read the option file. The name of the option file is gurobi.opt.

**ModelName.PriorOpt = 1;**
Instructs GAMS/Gurobi to use the priority branching information passed by GAMS through variable suffix values variable.prior.

5 Summary of GUROBI Options

5.1 Termination options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bariterlimit</td>
<td>Limits the number of barrier iterations performed</td>
<td>infinity</td>
</tr>
<tr>
<td>cutoff</td>
<td>Sets a target objective value</td>
<td>0</td>
</tr>
<tr>
<td>iterationlimit</td>
<td>Limits the number of simplex iterations performed</td>
<td>infinity</td>
</tr>
<tr>
<td>nodelimit</td>
<td>Limits the number of MIP nodes explored</td>
<td>maxint</td>
</tr>
<tr>
<td>solutionlimit</td>
<td>Limits the number of feasible solutions found</td>
<td></td>
</tr>
<tr>
<td>timelimit</td>
<td>Limits the total time expended in seconds</td>
<td>GAMS reslim</td>
</tr>
</tbody>
</table>

5.2 Tolerance options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>barconvtol</td>
<td>Controls barrier termination</td>
<td>1e-8</td>
</tr>
<tr>
<td>barqcpconvtol</td>
<td>Convergence tolerance for the barrier algorithm when solving a QCP</td>
<td>1e-6</td>
</tr>
<tr>
<td>feasibilitytol</td>
<td>Primal feasibility tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>intfeastol</td>
<td>Integer feasibility tolerance</td>
<td>1e-5</td>
</tr>
<tr>
<td>markowitztol</td>
<td>Threshold pivoting tolerance</td>
<td>0.0078125</td>
</tr>
<tr>
<td>mipgap</td>
<td>Relative MIP optimality gap</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td>mipgapabs</td>
<td>Absolute MIP optimality gap</td>
<td>GAMS optca</td>
</tr>
<tr>
<td>optimalitytol</td>
<td>Dual feasibility tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>psdtol</td>
<td>Limit on the amount of diagonal perturbation</td>
<td>1e-6</td>
</tr>
</tbody>
</table>

5.3 Simplex options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>normadjust</td>
<td>Pricing norm variants</td>
<td>-1</td>
</tr>
<tr>
<td>objscale</td>
<td>Objective coefficients scaling</td>
<td>0</td>
</tr>
<tr>
<td>perturbvalue</td>
<td>Magnitude of simplex perturbation when required</td>
<td>0.0002</td>
</tr>
<tr>
<td>quad</td>
<td>Quad precision computation in simplex</td>
<td>-1</td>
</tr>
<tr>
<td>scaleflag</td>
<td>Enables or disables model scaling</td>
<td>1</td>
</tr>
<tr>
<td>sifting</td>
<td>Sifting within dual simplex</td>
<td>-1</td>
</tr>
<tr>
<td>sifmethod</td>
<td>LP method used to solve sifting sub-problems</td>
<td>-1</td>
</tr>
<tr>
<td>simplexpricing</td>
<td>Determines variable pricing strategy</td>
<td>-1</td>
</tr>
</tbody>
</table>

5.4 Barrier options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>barcorrectors</td>
<td>Limits the number of central corrections performed in each barrier iteration</td>
<td>-1</td>
</tr>
<tr>
<td>barhomogeneous</td>
<td>Homogeneous barrier algorithm selection</td>
<td>-1</td>
</tr>
<tr>
<td>barorder</td>
<td>Chooses the barrier sparse matrix fill-reducing algorithm</td>
<td>-1</td>
</tr>
<tr>
<td>crossover</td>
<td>Determines the crossover strategy used to transform the barrier solution into a basic solution</td>
<td>-1</td>
</tr>
<tr>
<td>crossoverbasis</td>
<td>Determines the initial basis construction strategy for crossover</td>
<td>0</td>
</tr>
<tr>
<td>qcpdual</td>
<td>Determines whether dual variable values are computed for QCP models</td>
<td>1</td>
</tr>
</tbody>
</table>

5.5 MIP options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branchdir</td>
<td>Determines which child node is explored first in the branch-and-cut search</td>
<td>0</td>
</tr>
<tr>
<td>cliquecuts</td>
<td>Controls clique cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>concurrentmip</td>
<td>Enables the concurrent MIP solver</td>
<td>1</td>
</tr>
<tr>
<td>covercuts</td>
<td>Controls cover cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>cutaggpasses</td>
<td>Maximum number of aggregation passes during cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>cutpasses</td>
<td>Maximum number of cutting plane passes performed during root cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>cuts</td>
<td>Global cut generation control</td>
<td>-1</td>
</tr>
<tr>
<td>disconnected</td>
<td>Disconnected component strategy</td>
<td>-1</td>
</tr>
<tr>
<td>distributedmipjobs</td>
<td>Distributed MIP job count</td>
<td>0</td>
</tr>
<tr>
<td>flowcovercuts</td>
<td>Controls flow cover cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>flowpathcuts</td>
<td>Controls flow path cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>gomorypasses</td>
<td>Maximum number of Gomory cut passes</td>
<td>-1</td>
</tr>
<tr>
<td>gubcovercuts</td>
<td>Controls GUB cover cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics</td>
<td>Controls the amount of time spent in MIP heuristics</td>
<td>0.05</td>
</tr>
<tr>
<td>impliedcuts</td>
<td>Controls implied bound cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>improvestartgap</td>
<td>Optimality gap at which the MIP solver resets a few MIP parameters</td>
<td>maxdouble</td>
</tr>
<tr>
<td>improvestartnodes</td>
<td>Solution improvement strategy control</td>
<td>maxdouble</td>
</tr>
<tr>
<td>improvestarttime</td>
<td>Elapsed time after which the MIP solver resets a few MIP parameters</td>
<td>maxdouble</td>
</tr>
<tr>
<td>minrelnodes</td>
<td>Number of nodes to explore in the Minimum Relaxation heuristic</td>
<td>0</td>
</tr>
<tr>
<td>mipfocus</td>
<td>Controls the focus of the MIP solver</td>
<td>0</td>
</tr>
<tr>
<td>mipsepcuts</td>
<td>Controls MIP separation cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>miqcpmethod</td>
<td>Determines whether outer approximation is used to solve an MIQCP model.</td>
<td>-1</td>
</tr>
<tr>
<td>mircuts</td>
<td>Controls MIR cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>modkcuts</td>
<td>Controls the generation of mod-k cuts</td>
<td>-1</td>
</tr>
<tr>
<td>networkcuts</td>
<td>Controls network cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>nodefiledir</td>
<td>Nodedefile directory</td>
<td>.</td>
</tr>
<tr>
<td>nodefilestart</td>
<td>Nodedefile starting indicator</td>
<td>maxdouble</td>
</tr>
<tr>
<td>nodemethod</td>
<td>Algorithm used to solve node relaxations in a MIP model</td>
<td>1</td>
</tr>
<tr>
<td>norelheuristic</td>
<td>No relaxation heuristic attempts to find feasible solutions without solving root relaxation</td>
<td>0</td>
</tr>
<tr>
<td>presos1bigm</td>
<td>Threshold for SOS1-to-binary reformulation</td>
<td>-1</td>
</tr>
<tr>
<td>presos2bigm</td>
<td>Threshold for SOS2-to-binary reformulation</td>
<td>0</td>
</tr>
<tr>
<td>presparsify</td>
<td>Enables the presolve sparsify reduction for MIP models</td>
<td>0</td>
</tr>
<tr>
<td>.prior</td>
<td>Branching priorities</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>pumppasses</td>
<td>Number of passes of the feasibility pump heuristic</td>
<td>0</td>
</tr>
<tr>
<td>rins</td>
<td>Frequency of the RINS heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>submipcuts</td>
<td>Controls the generation of sub-MIP cutting planes</td>
<td>-1</td>
</tr>
<tr>
<td>submipnodes</td>
<td>Limits the number of nodes explored by the heuristics</td>
<td>500</td>
</tr>
<tr>
<td>symmetry</td>
<td>Controls MIP symmetry detection</td>
<td>-1</td>
</tr>
<tr>
<td>varbranch</td>
<td>Controls the branch variable selection strategy</td>
<td>-1</td>
</tr>
<tr>
<td>zerohalfcuts</td>
<td>Controls zero-half cut generation</td>
<td>-1</td>
</tr>
<tr>
<td>zeroobjnodes</td>
<td>Number of nodes to explore in the zero objective heuristic</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.6 Other options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggfill</td>
<td>Controls the amount of fill allowed during presolve aggregation</td>
<td>10</td>
</tr>
<tr>
<td>aggregate</td>
<td>Enables or disables aggregation in presolve</td>
<td>1</td>
</tr>
<tr>
<td>computeserver</td>
<td>List of Gurobi compute servers</td>
<td></td>
</tr>
<tr>
<td>concurrentjobs</td>
<td>Distributed concurrent MIP job count</td>
<td>0</td>
</tr>
<tr>
<td>cspassword</td>
<td>Password for Gurobi compute servers</td>
<td></td>
</tr>
<tr>
<td>csport</td>
<td>The port number used to connect to the compute server</td>
<td>-1</td>
</tr>
<tr>
<td>cspriority</td>
<td>Job priority on the compute server</td>
<td>0</td>
</tr>
<tr>
<td>cstimeoutfrac</td>
<td>Fraction of reslim that job will wait to reach front of the compute server queue</td>
<td>0.1</td>
</tr>
<tr>
<td>displayinterval</td>
<td>Controls the frequency at which log lines are printed in seconds</td>
<td>5</td>
</tr>
<tr>
<td>dumpsolution</td>
<td>Controls export of alternate MIP solutions</td>
<td></td>
</tr>
<tr>
<td>feasopt</td>
<td>Computes a minimum-cost relaxation to make an infeasible model feasible</td>
<td>0</td>
</tr>
<tr>
<td>feasoptmode</td>
<td>Mode of FeasOpt</td>
<td>0</td>
</tr>
<tr>
<td>.feaspref</td>
<td>feasibility preference</td>
<td>1</td>
</tr>
<tr>
<td>fixoptfile</td>
<td>Option file for fixed problem optimization</td>
<td></td>
</tr>
<tr>
<td>iis</td>
<td>Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>iismethod</td>
<td>Controls use of IIS method</td>
<td>-1</td>
</tr>
<tr>
<td>kappa</td>
<td>Display approximate condition number estimates for the optimal simplex basis</td>
<td>0</td>
</tr>
<tr>
<td>kappaexact</td>
<td>Display exact condition number estimates for the optimal simplex basis</td>
<td>0</td>
</tr>
<tr>
<td>.lazy</td>
<td>Lazy constraints value</td>
<td>0</td>
</tr>
<tr>
<td>lazyconstraints</td>
<td>Indicator to use lazy constraints</td>
<td>0</td>
</tr>
<tr>
<td>method</td>
<td>Algorithm used to solve continuous models</td>
<td>-1</td>
</tr>
<tr>
<td>mipstart</td>
<td>Use mip starting values</td>
<td>0</td>
</tr>
<tr>
<td>names</td>
<td>Indicator for loading names</td>
<td>1</td>
</tr>
<tr>
<td>numericfocus</td>
<td>Set the numerical focus</td>
<td>0</td>
</tr>
<tr>
<td>precrush</td>
<td>Presolve constraint option</td>
<td>0</td>
</tr>
<tr>
<td>predeprw</td>
<td>Controls the presolve dependent row reduction</td>
<td>-1</td>
</tr>
<tr>
<td>predual</td>
<td>Controls whether presolve forms the dual of a continuous model</td>
<td>-1</td>
</tr>
<tr>
<td>premiqmethod</td>
<td>Transformation presolve performs on MIQP models</td>
<td>-1</td>
</tr>
<tr>
<td>prepasses</td>
<td>Controls the number of passes performed by presolve</td>
<td>-1</td>
</tr>
<tr>
<td>prelinearize</td>
<td>Controls linearization of Q matrices in the quadratic constraints or a quadratic objective</td>
<td>-1</td>
</tr>
<tr>
<td>presolve</td>
<td>Controls the presolve level</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>printoptions</td>
<td>List values of all options to GAMS listing file</td>
<td>0</td>
</tr>
<tr>
<td>readparams</td>
<td>Read Gurobi parameter file</td>
<td></td>
</tr>
<tr>
<td>rerun</td>
<td>Resolve without presolve in case of unbounded or infeasible</td>
<td>0</td>
</tr>
<tr>
<td>seed</td>
<td>Random number seed</td>
<td>0</td>
</tr>
<tr>
<td>sensitivity</td>
<td>Provide sensitivity information</td>
<td>0</td>
</tr>
<tr>
<td>solvefixed</td>
<td>Indicator for solving the fixed problem for a MIP to get a dual solution</td>
<td>1</td>
</tr>
<tr>
<td>threads</td>
<td>Controls the number of threads to apply to parallel MIP or Barrier</td>
<td>GAMS threads</td>
</tr>
<tr>
<td>tunejobs</td>
<td>Distributed tuning job count</td>
<td></td>
</tr>
<tr>
<td>tuneoutput</td>
<td>Tuning output level</td>
<td>0</td>
</tr>
<tr>
<td>tuneresults</td>
<td>Number of improved parameter sets returned</td>
<td>1</td>
</tr>
<tr>
<td>tunetimelimit</td>
<td>Time limit for tuning</td>
<td>-1</td>
</tr>
<tr>
<td>tunetrials</td>
<td>Perform multiple runs on each parameter set to limit the effect of random noise</td>
<td>2</td>
</tr>
<tr>
<td>tuning</td>
<td>Parameter Tuning</td>
<td></td>
</tr>
<tr>
<td>usebasis</td>
<td>Use basis from GAMS</td>
<td>GAMS bratio</td>
</tr>
<tr>
<td>workerpassword</td>
<td>Compute server password Pool of compute servers to use for distributed algorithms</td>
<td></td>
</tr>
<tr>
<td>workerpool</td>
<td>Pool of compute servers to use for distributed algorithms</td>
<td></td>
</tr>
<tr>
<td>writeparams</td>
<td>Write Gurobi parameter file</td>
<td></td>
</tr>
<tr>
<td>writeprob</td>
<td>Save the problem instance</td>
<td></td>
</tr>
</tbody>
</table>

### 5.7 The GAMS/Gurobi Options File

The GAMS/Gurobi options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file `gurobi.opt`.

```
simplexpricing 3
method 0
```

It will cause Gurobi to use quick-start steepest edge pricing and will use the primal simplex algorithm.

### 6 GAMS/Gurobi Log File

Gurobi reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the simplex algorithms, each log line starts with the iteration number, followed by the objective value, the primal and dual infeasibility values, and the elapsed wall clock time. The dual simplex uses a bigM approach for handling infeasibility, so the objective and primal infeasibility values can both be very large during phase I. The frequency at which log lines are printed is controlled by the `displayinterval` option. By default, the simplex algorithms print a log line roughly every five seconds, although log lines can be delayed when solving models with particularly expensive iterations.

The simplex screen log has the following appearance:

```
Presolve removed 977 rows and 1539 columns
Presolve changed 3 inequalities to equalities
Presolve time: 0.078000 sec.
Presolved: 1748 Rows, 5030 Columns, 32973 Nonzeros
```
Iteration | Objective | Primal Inf. | Dual Inf. | Time
--- | --- | --- | --- | ---
0 | 3.8929476e+31 | 1.2000000e+31 | 1.485042e-04 | 0s
5624 | 1.1486966e+05 | 0.0000000e+00 | 0.0000000e+00 | 2s

Solved in 5624 iterations and 1.69 seconds
Optimal objective 1.148696610e+05

The barrier algorithm log file starts with barrier statistics about dense columns, free variables, nonzeros in AA’ and the Cholesky factor matrix, computational operations needed for the factorization, memory estimate and time estimate per iteration. Then it outputs the progress of the barrier algorithm in iterations with the primal and dual objective values, the magnitude of the primal and dual infeasibilities and the magnitude of the complementarity violation. After the barrier algorithm terminates, by default, Gurobi will perform crossover to obtain a valid basic solution. It first prints the information about pushing the dual and primal superbasic variables to the bounds and then the information about the simplex progress until the completion of the optimization.

The barrier screen log has the following appearance:

Presolve removed 2394 rows and 3412 columns
Presolve time: 0.09s
Presolved: 3677 Rows, 8818 Columns, 30934 Nonzeros

Ordering time: 0.20s

Barrier statistics:
Dense cols : 10
Free vars : 3
AA’ NZ : 9.353e+04
Factor NZ : 1.139e+06 (roughly 14 MBytes of memory)
Factor Ops : 7.388e+08 (roughly 2 seconds per iteration)

<table>
<thead>
<tr>
<th>Iter</th>
<th>Primal</th>
<th>Dual</th>
<th>Primal</th>
<th>Dual</th>
<th>Compl</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.11502515e+13</td>
<td>-3.03102251e+08</td>
<td>7.65e+05</td>
<td>9.29e+07</td>
<td>2.68e+09</td>
<td>2s</td>
</tr>
<tr>
<td>1</td>
<td>4.40823949e+12</td>
<td>-8.22101865e+09</td>
<td>3.10e+05</td>
<td>4.82e+07</td>
<td>1.15e+09</td>
<td>3s</td>
</tr>
<tr>
<td>2</td>
<td>1.18016996e+12</td>
<td>-2.50595257e+10</td>
<td>7.39e+04</td>
<td>1.15e+07</td>
<td>3.37e+08</td>
<td>4s</td>
</tr>
<tr>
<td>3</td>
<td>2.24969338e+11</td>
<td>-2.9167762e+10</td>
<td>1.01e+04</td>
<td>2.16e+06</td>
<td>5.51e+07</td>
<td>5s</td>
</tr>
<tr>
<td>4</td>
<td>4.63363675e+10</td>
<td>-1.4430875e+10</td>
<td>8.13e+02</td>
<td>4.30e+05</td>
<td>9.09e+06</td>
<td>6s</td>
</tr>
<tr>
<td>5</td>
<td>1.25266057e+10</td>
<td>-4.06364070e+09</td>
<td>1.52e+02</td>
<td>8.13e+04</td>
<td>2.21e+06</td>
<td>7s</td>
</tr>
<tr>
<td>6</td>
<td>1.53128732e+09</td>
<td>-1.27023188e+09</td>
<td>9.52e+00</td>
<td>1.61e+04</td>
<td>3.23e+05</td>
<td>9s</td>
</tr>
<tr>
<td>7</td>
<td>5.70973983e+08</td>
<td>-8.11694302e+08</td>
<td>2.10e+00</td>
<td>5.99e+03</td>
<td>1.53e+05</td>
<td>10s</td>
</tr>
<tr>
<td>8</td>
<td>2.91659869e+08</td>
<td>-4.77256823e+08</td>
<td>5.89e-01</td>
<td>5.96e-08</td>
<td>8.36e+04</td>
<td>11s</td>
</tr>
<tr>
<td>9</td>
<td>1.22358325e+08</td>
<td>-1.30263121e+08</td>
<td>6.09e-02</td>
<td>7.36e-07</td>
<td>2.73e+04</td>
<td>12s</td>
</tr>
<tr>
<td>10</td>
<td>6.47115867e+07</td>
<td>-4.50505785e+07</td>
<td>1.96e-02</td>
<td>1.43e-06</td>
<td>1.18e+04</td>
<td>13s</td>
</tr>
<tr>
<td>.....</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1.12663966e+07</td>
<td>1.12663950e+07</td>
<td>1.85e-07</td>
<td>2.82e-06</td>
<td>1.74e-04</td>
<td>2s</td>
</tr>
<tr>
<td>27</td>
<td>1.12663961e+07</td>
<td>1.12663960e+07</td>
<td>3.87e-08</td>
<td>2.02e-07</td>
<td>8.46e-06</td>
<td>2s</td>
</tr>
</tbody>
</table>

Barrier solved model in 27 iterations and 1.86 seconds
Optimal objective 1.12663961e+07

Crossover log...

1592 DPushes remaining with DInf 0.0000000e+00
0 DPushes remaining with DInf 2.8167333e-06
180 PPushes remaining with PInf 0.0000000e+00
Gurobi 6.0

0 PPushes remaining with PInf 0.0000000e+00 2s

Push phase complete: Pinf 0.0000000e+00, Dinf 2.8167333e-06 2s

Iteration Objective Primal Inf. Dual Inf. Time
1776 1.1266396e+07 0.0000000e+00 0.0000000e+00 2s

Solved in 2043 iterations and 2.00 seconds
Optimal objective 1.126639605e+07

For MIP problems, the Gurobi solver prints regular status information during the branch and bound search. The first two output columns in each log line show the number of nodes that have been explored so far in the search tree, followed by the number of nodes that remain unexplored. The next three columns provide information on the most recently explored node in the tree. The solver prints the relaxation objective value for this node, followed by its depth in the search tree, followed by the number of integer variables with fractional values in the node relaxation solution. The next three columns provide information on the progress of the global MIP bounds. They show the objective value for the best known integer feasible solution, the best bound on the value of the optimal solution, and the gap between these lower and upper bounds. Finally, the last two columns provide information on the amount of work performed so far. The first column gives the average number of simplex iterations per explored node, and the next column gives the elapsed wall clock time since the optimization began.

At the default value for option `displayinterval`, the MIP solver prints one log line roughly every five seconds. Note, however, that log lines are often delayed in the MIP solver due to particularly expensive nodes or heuristics.

Presolve removed 12 rows and 11 columns
Presolve tightened 70 bounds and modified 235 coefficients
Presolve time: 0.02s
Presolved: 114 Rows, 116 Columns, 424 Nonzeros
Objective GCD is 1

Nodes | Current Node | Objective Bounds | Work
Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time
H 0 0 -0.0000 - - - 0s
Root relaxation: 208 iterations, 0.00 seconds
0 0 29.6862 0 64 -0.0000 29.6862 - - 0s
H 0 0 8.0000 29.6862 271% - 0s
H 0 0 17.0000 29.6862 74.6% - 0s
0 2 27.4079 0 60 17.0000 27.4079 61.2% - 0s
H 27 17 18.0000 26.0300 44.6% 51.6 0s
* 87 26 18.0000 26.0300 44.6% 51.6 0s
* 353 71 20.0000 26.0300 30.2% 28.4 0s
1268 225 24.0000 28 43 21.0000 24.0000 14.3% 32.3 5s
2215 464 22.0000 43 30 21.0000 24.0000 14.3% 33.2 10s

Cutting planes:
Gomory: 175
Cover: 25
Implied bound: 87
MIR: 150

Explored 2550 nodes (84600 simplex iterations) in 11.67 seconds
Thread count was 1 (of 4 available processors)

Optimal solution found (tolerance 1.00e-01)
Best objective 2.1000000000e+01, best bound 2.3000000000e+01, gap 9.5238%
7 Detailed Descriptions of GUROBI Options

**aggfill (integer):** Controls the amount of fill allowed during presolve aggregation

Larger values generally lead to presolved models with fewer rows and columns, but with more constraint matrix non-zeros.

(default = 10)

**aggregate (integer):** Enables or disables aggregation in presolve

(default = 1)

**barconvtol (real):** Controls barrier termination

The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance.

(default = 1e-8)

**barcorrectors (integer):** Limits the number of central corrections performed in each barrier iteration

The default value is choosen automatically, depending on problem characteristics.

(default = -1)

**barhomogeneous (integer):** Homogeneous barrier algorithm selection

Determines whether to use the homogeneous barrier algorithm. At the default setting (-1), it is only used when barrier solves a node relaxation for a MIP model. Setting the parameter to 0 turns it off, and setting it to 1 forces it on. The homogeneous algorithm is useful for recognizing infeasibility or unboundedness. It is a bit slower than the default algorithm.

(default = -1)

-1 Auto
0 Homogeneous Barrier off
1 Force Homogeneous Barrier on

**bariterlimit (integer):** Limits the number of barrier iterations performed

(default = infinity)

**barorder (integer):** Chooses the barrier sparse matrix fill-reducing algorithm

(default = -1)

-1 Auto
0 Approximate Minimum Degree ordering
1 Nested Dissection ordering

**barqcpconvtol (real):** Convergence tolerance for the barrier algorithm when solving a QCP

When solving a QCP model, the barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. Tightening this tolerance may lead to a more accurate solution, but it may also lead to a failure to converge.

(default = 1e-6)

**branchdir (integer):** Determines which child node is explored first in the branch-and-cut search

This option allows more control over how the branch-and-cut tree is explored. Specifically, when a node in the MIP search is completed and two child nodes, corresponding to the down branch and the up branch are created, this parameter allows you to determine whether the MIP solver will explore the down branch first, the up branch first, or whether it will choose the next node based on a heuristic determination of which sub-tree appears more promising.

(default = 0)
-1 Always explore the down branch first
0 Automatic
1 Always explore the up branch first

cliquecuts (integer): Controls clique cut generation
See the description of the global Cuts parameter for further information.
(default = -1)
  -1 Auto
  0 Off
  1 Conservative
  2 Aggressive

computeserver (string): List of Gurobi compute servers
This option will use the Gurobi compute server facility to solve the model. The value of the option is a comma separated list of IP addresses or computer names of Gurobi compute servers. On the compute servers, a full Gurobi installation (not just GAMS/Gurobi) is required with an appropriate Gurobi compute server license.

concurrentjobs (integer): Distributed concurrent MIP job count
Enables distributed concurrent optimization, which can be used to solve LP or MIP models on multiple machines. A value of \( n \) causes the solver to create \( n \) independent models, using different parameter settings for each. Each of these models is sent to a distributed worker for processing. Optimization terminates when the first solve completes. Use the WorkerPool parameter to provide a list of available distributed workers.

By default, Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search. By bringing the resources of multiple machines to bear on a single model, this approach can sometimes solve models much faster than a single machine.

(default = 0)

concurrentmip (integer): Enables the concurrent MIP solver
This parameter enables the concurrent MIP solver. When the parameter is set to value \( n \), the MIP solver performs \( n \) independent MIP solves in parallel, with different parameter settings for each. Optimization terminates when the first solve completes. Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search. This approach can sometimes solve models much faster than applying all available threads to a single MIP solve, especially on very large parallel machines.

The concurrent MIP solver divides available threads evenly among the independent solves. For example, if you have 6 threads available and you set ConcurrentMIP to 2, the concurrent MIP solver will allocate 3 threads to each independent solve. Note that the number of independent solves launched will not exceed the number of available threads.

The concurrent MIP solver produces a slightly different log from the standard MIP solver. The log only provides periodic summary information. Each concurrent MIP log line shows the objective for the best feasible solution found by any of the independent solves to that point, the best objective bound proved by any of the independent solves, and the relative gap between these two values. Gurobi also includes node counts from one of the independent solves, as well as elapsed times, to give some indication of forward progress.

(default = 1)

covercuts (integer): Controls cover cut generation
See the description of the global Cuts parameter for further information.
(default = -1)
  -1 Auto
  0 Off
1 Conservative
2 Aggressive
crossover (integer): Determines the crossover strategy used to transform the barrier solution into a basic solution
Use value 0 to disable crossover; the solver will return an interior solution. Other options control whether the
crossover algorithm tries to push primal or dual variables to bounds first, and then which simplex algorithm
is used once variable pushing is complete. Options 1 and 2 push dual variables first, then primal variables.
Option 1 finishes with primal, while option 2 finishes with dual. Options 3 and 4 push primal variables first, then
dual variables. Option 3 finishes with primal, while option 4 finishes with dual. The default value of -1 chooses
automatically.
(default = -1)
crossoverbasis (integer): Determines the initial basis construction strategy for crossover
The default value (0) chooses an initial basis quickly. A value of 1 can take much longer, but often produces a
much more numerically stable start basis.
(default = 0)
cspassword (string): Password for Gurobi compute servers
csport (integer): The port number used to connect to the compute server
You should use the default value, which indicates that the default port should be used, unless your server
administrator has changed our recommended port settings.
(default = -1)
cspriority (integer): Job priority on the compute server
Gurobi compute servers support job priorities. You can assign an integer priority between -100 and 100 to each
job (the default is 0). When choosing among queued jobs, the compute server will run the highest priority job
first. Note that servers will never preempt running jobs.
We have chosen to give priority 100 a special meaning. A priority 100 job will start immediately, even if this
means that the server will exceed its job limit. You should be cautious with priority 100 jobs, since submitting too
many at once could lead to very high server loads, which could lead to poor performance and even crashes in
extreme cases.
(default = 0)
cstimeoutfrac (real): Fraction of reslim that job will wait to reach front of the compute server queue
If the compute servers are all busy, new jobs will be queued. This option controls the fraction of the available
time (ResLim) that one is willing to wait in the queue before the optimization can start. This waiting time does
not count towards the actual time available to the optimizer.
(default = 0.1)
cutaggpasses (integer): Maximum number of aggregation passes during cut generation
A non-negative value indicates the maximum number of constraint aggregation passes performed during cut
geneneration. See the description of the global Cuts parameter for further information.
(default = -1)
cutoff (real): Sets a target objective value
Optimization will terminate if the engine determines that the optimal objective value for the model is worse than
the specified cutoff. This option overwrites the GAMS cutoff option.
(default = 0)
cutpasses (integer): Maximum number of cutting plane passes performed during root cut generation
(default = -1)
cuts (integer): Global cut generation control
The parameters, `cuts`, `CliqueCuts`, `CoverCuts`, `FlowCoverCuts`, `FlowPathCuts`, `GUBCoverCuts`, `ImpliedCuts`, `MIPSepCuts`, `MIRCuts`, `ModKCuts`, `NetworkCuts`, `GomoryPasses`, `SubMIPCuts`, `CutAggPasses` and `ZeroHalfCuts`, affect the generation of MIP cutting planes. In all cases except `gomorypasses` and `cutaggpasses`, a value of `-1` corresponds to an automatic setting, which allows the solver to determine the appropriate level of aggressiveness in the cut generation. Unless otherwise noted, settings of 0, 1, and 2 correspond to no cut generation, conservative cut generation, or aggressive cut generation, respectively. The `Cuts` parameter provides global cut control, affecting the generation of all cuts. This parameter also has a setting of 3, which corresponds to very aggressive cut generation. The other parameters override the global `Cuts` parameter (so setting `Cuts` to 2 and `CliqueCuts` to 0 would generate all cut types aggressively, except clique cuts which would not be generated at all. Setting `Cuts` to 0 and `GomoryPasses` to 10 would not generate any cuts except Gomory cuts for 10 passes).

(default = -1)

- 1 Auto
- 0 Off
- 1 Conservative
- 2 Aggressive
- 3 Very aggressive

**disconnected (integer):** Disconnected component strategy

A MIP model can sometimes be made up of multiple, completely independent sub-models. This parameter controls how aggressively we try to exploit this structure. A value of 0 ignores this structure entirely, while larger values try more aggressive approaches. The default value of -1 chooses automatically. This only affects mixed integer programming (MIP) models.

(default = -1)

- 1 Auto
- 0 Ignores structure entirely
- 1 Conservative
- 2 Aggressive

**displayinterval (integer):** Controls the frequency at which log lines are printed in seconds

(default = 5)

**distributedmipjobs (integer):** Distributed MIP job count

Enables distributed MIP. A value of \( n \) causes the MIP solver to divide the work of solving a MIP model among \( n \) machines. Use the `WorkerPool` parameter to provide the list of available machines.

(default = 0)

**dumpsolution (string):** Controls export of alternate MIP solutions

The GDX file specified by this option will contain a set call `index` that contains the names of GDX files with the individual solutions. For details see example model `dumpsol` in the GAMS Test Library.

**feasibilitytol (real):** Primal feasibility tolerance

All constrains must be satisfied to a tolerance of `FeasibilityTol`.

Range: \([1e-9, 1e-2]\)

(default = 1e-6)

**feaso (integer):** Computes a minimum-cost relaxation to make an infeasible model feasible

With `Feaso` turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option `FeasOptMode`. 
Feasible relaxations are available for all problem types.

(default = 0)

0 Turns Feasible Relaxation off
1 Turns Feasible Relaxation on

feasoptmode (integer): Mode of FeasOpt

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a sum of required relaxations; as the number of constraints and bounds required to be relaxed; as a sum of the squares of required relaxations).

(default = 0)

0 Minimize sum of relaxations Minimize the sum of all required relaxations in first phase only
1 Minimize sum of relaxations and optimize Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations
2 Minimize number of relaxations Minimize the number of constraints and bounds requiring relaxation in first phase only
3 Minimize number of relaxations and optimize Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations
4 Minimize sum of squares of relaxations Minimize the sum of squares of required relaxations in first phase only
5 Minimize sum of squares of relaxations and optimize Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref (real): feasibility preference

You can express the costs associated with relaxing a bound or right hand side value during a FeasOpt run through the .feaspref option. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

(default = 1)

fixoptfile (string): Option file for fixed problem optimization

flowcovercuts (integer): Controls flow cover cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

-1 Auto
0 Off
1 Conservative
2 Aggressive

flowpathcuts (integer): Controls flow path cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

-1 Auto
0 Off
1 Conservative
2 Aggressive

gomorypasses (integer): Maximum number of Gomory cut passes
A non-negative value indicates the maximum number of Gomory cut passes performed. See the description of the global Cuts parameter for further information.
(default = -1)

gubcovercuts (integer): Controls GUB cover cut generation
See the description of the global Cuts parameter for further information.
(default = -1)

heuristics (real): Controls the amount of time spent in MIP heuristics
Larger values produce more and better feasible solutions, at a cost of slower progress in the best bound.
Range: [0, 1]
(default = 0.05)

iis (integer): Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible
(default = 0)
0 Do not run the IIS finder
1 Run the IIS finder

iismethod (integer): Controls use of IIS method
Chooses the IIS method to use. Method 0 is often faster, while method 1 can produce a smaller IIS. The default value of -1 chooses automatically.
(default = -1)

impliedcuts (integer): Controls implied bound cut generation
See the description of the global Cuts parameter for further information.
(default = -1)

improvestartgap (real): Optimality gap at which the MIP solver resets a few MIP parameters
The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify an optimality gap at which the MIP solver will switch to this strategy. For example, setting this parameter to 0.1 will cause the MIP solver to switch once the relative optimality gap is smaller than 0.1.
(default = maxdouble)

improvestartnodes (real): Solution improvement strategy control
The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify the node count at which the MIP solver switches to a solution improvement strategy. For example, setting this parameter to 10 will cause the MIP solver to switch strategies once the node count is larger than 10.

(default = maxdouble)

`improvestarttime (real)`: Elapsed time after which the MIP solver resets a few MIP parameters

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify a time limit when the MIP solver will switch to this strategy. For example, setting this parameter to 10 will cause the MIP solver to switch 10 seconds after starting the optimization.

(default = maxdouble)

`intfeastol (real)`: Integer feasibility tolerance

An integrality restriction on a variable is considered satisfied when the variable’s value is less than $\text{IntFeasTol}$ from the nearest integer value.

Range: $[1e-9, 1e-1]$  
(default = $1e-5$)

`iterationlimit (real)`: Limits the number of simplex iterations performed

(default = $\text{infinity}$)

`kappa (integer)`: Display approximate condition number estimates for the optimal simplex basis

(default = 0)

- 0 Do not compute and display approximate condition number
- 1 Compute and display approximate condition number

`kappaexact (integer)`: Display exact condition number estimates for the optimal simplex basis

(default = 0)

- 0 Do not compute and display exact condition number
- 1 Compute and display exact condition number

`lazy (integer)`: Lazy constraints value

Determines whether a linear constraint is treated as a lazy constraint. At the beginning of the MIP solution process, any constraint whose Lazy attribute is set to 1, 2, or 3 (the default value is 0) is removed from the model and placed in the lazy constraint pool. Lazy constraints remain inactive until a feasible solution is found, at which point the solution is checked against the lazy constraint pool. If the solution violates any lazy constraints, the solution is discarded and one of more of the violated lazy constraints are pulled into the active model.

Larger values for this attribute cause the constraint to be pulled into the model more aggressively. With a value of 1, the constraint can be used to cut off a feasible solution, but it won’t necessarily be pulled in if another lazy constraint also cuts off the solution. With a value of 2, all lazy constraints that are violated by a feasible solution will be pulled into the model. With a value of 3, lazy constraints that cut off the relaxation solution are also pulled in.

Only affects MIP models. Lazy constraints are only active if option `LazyConstraints` is enabled and are specified through the option `.lazy`.

(default = 0)

`lazyconstraints (integer)`: Indicator to use lazy constraints

(default = 0)

`markowitztol (real)`: Threshold pivoting tolerance

Used to limit numerical error in the simplex algorithm. A larger value may avoid numerical problems in rare
situations, but it will also harm performance.
Range: \([1e-4, 0.999]\]
(default = 0.0078125)

**method (integer):** Algorithm used to solve continuous models

Synonyms: lpmethod rootmethod

Concurrent optimizers run multiple solvers on multiple threads simultaneously, and choose the one that finishes first. Deterministic concurrent (4) gives the exact same result each time, while concurrent (3) is often faster but can produce different optimal bases when run multiple times. In the current release, the default Automatic (-1) will choose non-deterministic concurrent (3) for an LP, barrier (2) for a QP, and dual (1) for the MIP root node. Only simplex and barrier algorithms are available for continuous QP models. Only primal and dual simplex are available for solving the root of an MIQP model. Only barrier is available for continuous QCP models.

The default setting is rarely significantly slower than the best possible setting, so you generally won’t see a big gain from changing this parameter. There are classes of models where one particular algorithm is consistently fastest, though, so you may want to experiment with different options when confronted with a particularly difficult model.

Note that if memory is tight on an LP model, you should consider choosing the dual simplex method (Method=1). The default will invoke the concurrent optimizer, which typically consumes a lot more memory than dual simplex alone.

(default = -1)

-1 Automatic
0 Primal simplex
1 Dual simplex
2 Barrier
3 Concurrent
4 Deterministic concurrent

**minrelnodes (integer):** Number of nodes to explore in the Minimum Relaxation heuristic

This parameter controls the Minimum Relaxation heuristic that can be useful for finding solutions to MIP models where other strategies fail to find feasible solutions in a reasonable amount of time. This heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

(default = 0)

**mipfocus (integer):** Controls the focus of the MIP solver

(default = 0)

0 Balance between finding good feasible solutions and proving optimality
1 Focus towards finding feasible solutions
2 Focus towards proving optimality
3 Focus on moving the best objective bound

**mipgap (real):** Relative MIP optimality gap

The MIP engine will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than MipGap times the upper bound.

Range: \([0, \text{maxdouble}]\]

(default = GAMS optcr)

**mipgapabs (real):** Absolute MIP optimality gap

The MIP solver will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than MIPGapAbs.
Range: [0, maxdouble]
(default = GAMS optca)

**mipsepcuts (integer):** Controls MIP separation cut generation

See the description of the global Cuts parameter for further information.

(default = -1)
-1 Auto
0 Off
1 Conservative
2 Aggressive

**mipstart (integer):** Use mip starting values

(default = 0)
0 Do not use the values
1 Use the values

**miqcpmethod (integer):** Determines whether outer approximation is used to solve an MIQCP model.

Controls the method used to solve MIQCP models. Value 1 uses a linearized, outer-approximation approach, while value 0 solves continuous QCP relaxations at each node. The default setting (-1) chooses automatically.

(default = -1)
-1 Auto
0 Linearized, outer-approximation approach
1 Continuous QCP relaxations at each node

**mircuts (integer):** Controls MIR cut generation

See the description of the global Cuts parameter for further information.

(default = -1)
-1 Auto
0 Off
1 Conservative
2 Aggressive

**modkcuts (integer):** Controls the generation of mod-k cuts

See the description of the global Cuts parameter for further information.

(default = -1)

**names (integer):** Indicator for loading names

(default = 1)
0 Do not load GAMS names into Gurobi model
1 Load GAMS names into Gurobi model

**networkcuts (integer):** Controls network cut generation

See the description of the global Cuts parameter for further information.

(default = -1)
-1 Auto
0 Off
1 Conservative
2 Aggressive

**nodefiledir** *(string)*: Nodefile directory

Determines the directory into which nodes are written when node memory usage exceeds the specified NodefileStart value.

(default = .)

**nodefilestart** *(real)*: Nodefile starting indicator

Controls the point at which MIP tree nodes are written to disk. Whenever node storage exceeds the specified value (in GBytes), nodes are written to disk.

(default = maxdouble)

**nodelimit** *(real)*: Limits the number of MIP nodes explored

(default = maxdouble)

**nodemethod** *(integer)*: Algorithm used to solve node relaxations in a MIP model

Algorithm used for MIP node relaxations. Note that barrier is not an option for MIQP node relaxations.

(default = 1)

0 Primal simplex
1 Dual simplex
2 Barrier

**norealheuristic** *(integer)*: No relaxation heuristic attempts to find feasible solutions without solving root relaxation

Gurobi 5.6.2 introduces an experimental no relaxation heuristic that attempts to find good quality feasible solutions to MIP models without first solving the root relaxation. This can be useful in situations where the root relaxation is extremely large or difficult.

(default = 0)

0 Do not use no relaxation heuristic
1 Try no relaxation heuristic

**normadjust** *(integer)*: Pricing norm variants

Chooses from among multiple pricing norm variants. The default value of -1 chooses automatically.

(default = -1)

**numericfocus** *(integer)*: Set the numerical focus

The NumericFocus parameter controls the degree to which the code attempts to detect and manage numerical issues. The default setting makes an automatic choice, with a slight preference for speed. Settings 1-3 increasingly shift the focus towards being more careful in numerical computations. With higher values, the code will spend more time checking the numerical accuracy of intermediate results, and it will employ more expensive techniques in order to avoid potential numerical issues.

(default = 0)

**objcscale** *(real)*: Objective coefficients scaling

Divides the model objective by the specified value to avoid numerical errors that may result from very large objective coefficients. The default value of 0 decides on the scaling automatically. A value less than zero uses the maximum coefficient to the specified power as the scaling (so ObjScale=-0.5 would scale by the square root of the largest objective coefficient).

Range: [-1, maxdouble]

(default = 0)

**optimalitytol** *(real)*: Dual feasibility tolerance

Reduced costs must all be larger than OptimalityTol in the improving direction in order for a model to be declared
optimal.
Range: $[1e-9, 1e-2]$
(default = $1e-6$)

**perturbvalue (real):** Magnitude of simplex perturbation when required
Range: $[0, 0.01]$
(default = 0.0002)

**precrush (integer):** Presolve constraint option
Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model.
This parameter is turned on when you use BCH with Gurobi.
(default = 0)

**predeprrow (integer):** Controls the presolve dependent row reduction
Controls the presolve dependent row reduction, which eliminates linearly dependent constraints from the constraint matrix. The default setting (-1) applies the reduction to continuous models but not to MIP models. Setting 0 turns the reduction off for all models. Setting 1 turns it on for all models.
(default = -1)

**predual (integer):** Controls whether presolve forms the dual of a continuous model
Depending on the structure of the model, solving the dual can reduce overall solution time. The default setting uses a heuristic to decide. Setting 0 forbids presolve from forming the dual, while setting 1 forces it to take the dual. Setting 2 employs a more expensive heuristic that forms both the presolved primal and dual models (on two threads), and heuristically chooses one of them.
(default = -1)

**premiqpmethod (integer):** Transformation presolve performs on MIQP models
Chooses the transformation presolve performs on MIQP models.
(default = -1)

-1 Auto
0 Always leaves the model as an MIQP
1 Attempts to transform the model into an MILP

**prepasses (integer):** Controls the number of passes performed by presolve
Limits the number of passes performed by presolve. The default setting (-1) chooses the number of passes automatically.
(default = -1)

**preqlinearize (integer):** Controls linearization of Q matrices in the quadratic constraints or a quadratic objective
Option 1 attempts to linearize quadratic constraints or a quadratic objective, potentially transforming an MIQP or MIQCP into an MILP. Option 0 shuts off the transformation. The default setting (-1) choose automatically. The automatic setting works well, but there are cases where forcing Q linearization can be beneficial.
(default = -1)

-1 Auto
0 Linearization off
1 Force Linearization on

**presolve (integer):** Controls the presolve level
(default = -1)
-1 Auto
0 Off
1 Conservative
2 Aggressive

**presos1bigm (integer):** Threshold for SOS1-to-binary reformulation

Controls the automatic reformulation of SOS1 constraints into binary form. SOS1 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values increase the chances that an SOS1 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of -1 chooses a threshold automatically. You should set the parameter to 0 to shut off SOS1 reformulation entirely, or a large value to force reformulation.

Range: [-1, \text{maxint}]

(default = -1)

**presos2bigm (integer):** Threshold for SOS2-to-binary reformulation

Controls the automatic reformulation of SOS2 constraints into binary form. SOS2 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values increase the chances that an SOS2 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of 0 disables the reformulation. You can set the parameter to -1 to choose an automatic approach, or a large value to force reformulation.

Range: [-1, \text{maxint}]

(default = 0)

**presparsify (integer):** Enables the presolve sparsify reduction for MIP models

This reduction can sometimes significantly reduce the number of nonzero values in the presolved model.

(default = 0)

0 Disable the presolve sparsify reduction
1 Enable the presolve sparsify reduction

**printoptions (integer):** List values of all options to GAMS listing file

(default = 0)

0 Do not list option values to GAMS listing file
1 List option values to GAMS listing file

**.prior (real):** Branching priorities

GAMS allows to specify priorities for discrete variables only. Gurobi can detect that continuous variables are implied discrete variables and can utilize priorities. Such priorities can be specified through a GAMS/Gurobi solver option file. The syntax for dot options is explained in the Introduction chapter of the Solver Manual. The priorities are only passed on to Gurobi if the model attribute priorOpt is turned on.

(default = 1)

**psdtol (real):** limit on the amount of diagonal perturbation

Positive semi-definite tolerance (for QP/MIQP). Sets a limit on the amount of diagonal perturbation that the optimizer is allowed to automatically perform on the Q matrix in order to correct minor PSD violations. If a larger perturbation is required, the optimizer will terminate stating the problem is not PSD.

Range: [0, \text{maxdouble}]

(default = 1e-6)
**pumppasses (integer):** Number of passes of the feasibility pump heuristic

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic found a feasible solution.

(default = 0)

**qcpdual (integer):** Determines whether dual variable values are computed for QCP models

Determines whether dual variable values are computed for QCP models. Computing them can add significant time to the optimization, so you should turn this parameter to 0 if you do not need them.

(default = 1)

0 Compute dual for QCP problem
1 Do not compute dual for QCP problem

**quad (integer):** Quad precision computation in simplex

Enables or disables quad precision computation in simplex. The -1 default setting allows the algorithm to decide.

(default = -1)

**readparams (string):** Read Gurobi parameter file

**rerun (integer):** Resolve without presolve in case of unbounded or infeasible

In case Gurobi reports *Model was proven to be either infeasible or unbounded*, this option decides about a resolve without presolve which will determine the exact model status. If the option is set to *auto*, which is the default, and the model fits into demo limits, the problems is resolved.

(default = 0)

-1 No
0 Auto
1 Yes

**rins (integer):** Frequency of the RINS heuristic

Default value (-1) chooses automatically. A value of 0 shuts off RINS. A positive value \( n \) applies RINS at every \( n \)-th node of the MIP search tree.

(default = -1)

**scaleflag (integer):** Enables or disables model scaling

Controls model scaling. By default, the rows and columns of the model are scaled in order to improve the numerical properties of the constraint matrix. The scaling is removed before the final solution is returned. Scaling typically reduces solution times, but it may lead to larger constraint violations in the original, unscaled model. Turning off scaling with \&ScaleFlag=0\ can sometimes produce smaller constraint violations. Choosing a more aggressive scaling option with \&ScaleFlag=2\ can sometimes improve performance for particularly numerically difficult models.

(default = 1)

**seed (integer):** Random number seed

Modifies the random number seed. This acts as a small perturbation to the solver, and typically leads to different solution paths.

(default = 0)

**sensitivity (integer):** Provide sensitivity information

(default = 0)

0 Do not provide sensitivity information
1 Provide sensitivity information

**sifting (integer):** Sifting within dual simplex
Enables or disables sifting within dual simplex. Sifting is often useful for LP models where the number of variables is many times larger than the number of constraints. With a Moderate setting, sifting will be applied to LP models and to the root node for MIP models. With an Aggressive setting, sifting will be also applied to the nodes of a MIP. Note that this parameter has no effect if you aren’t using dual simplex. Note also that sifting will be skipped in cases where it is obviously a worse choice, even when sifting has been selected.

(default = -1)

-1 Auto

0 Off

1 Moderate

2 Aggressive

**siftmethod** (integer): LP method used to solve sifting sub-problems

Note that this parameter only has an effect when you are using dual simplex and sifting has been selected (either by the automatic method, or through the Sifting parameter).

(default = -1)

-1 Auto

0 Primal Simplex

1 Dual Simplex

2 Barrier

**simplexpricing** (integer): Determines variable pricing strategy

(default = -1)

-1 Auto

0 Partial Pricing

1 Steepest Edge

2 Devex

3 Quick-Start Steepest Edge

**solutionlimit** (integer): Limits the number of feasible solutions found

(default = maxint)

**solvefixed** (integer): Indicator for solving the fixed problem for a MIP to get a dual solution

(default = 1)

0 Do not solve the fixed problem

1 Solve the fixed problem

**submipcuts** (integer): Controls the generation of sub-MIP cutting planes

See the description of the global Cuts parameter for further information.

(default = -1)

**submipnodes** (integer): Limits the number of nodes explored by the heuristics

Limits the number of nodes explored by the heuristics, like RINS. Exploring more nodes can produce better solutions, but it generally takes longer.

(default = 500)

**symmetry** (integer): Controls MIP symmetry detection

(default = -1)

-1 Auto
threads (integer): Controls the number of threads to apply to parallel MIP or Barrier

Default number of parallel threads allowed for any solution method. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

(tdefault = GAMS threads)

timelimit (real): Limits the total time expended in seconds

(tdefault = GAMS reslim)

tunejobs (integer): Distributed tuning job count

Enables distributed parallel tuning, which can significantly increase the performance of the tuning tool. A value of \( n \) causes the tuning tool to distribute tuning work among \( n \) parallel jobs. These jobs are distributed among a set of workers. Use the WorkerPool parameter to provide a list of available workers.

Note that distributed tuning is most effective when the workers have similar performance. Distributed tuning doesn’t attempt to normalize performance by worker, so it can incorrectly attribute a boost in performance to a parameter change when the associated setting is tried on a worker that is significantly faster than the others.

(tdefault = 0)

tuneoutput (integer): Tuning output level

(tdefault = 2)

0 No output
1 Summary output only when a new best parameter set is found
2 Summary output for each parameter set that is tried
3 Summary output, plus detailed solver output, for each parameter set tried

tuneresults (integer): Number of improved parameter sets returned

The tuning tool often finds multiple parameter sets that produce better results than the baseline settings. This parameter controls how many of these sets should be retained when tuning is complete.

(tdefault = 1)

tunetimelimit (real): Time limit for tuning

Limits total tuning runtime (in seconds). The default setting (-1) chooses a time limit automatically.

(tdefault = -1)

tunetrials (integer): Perform multiple runs on each parameter set to limit the effect of random noise

Performance on a MIP model can sometimes experience significant variations due to random effects. As a result, the tuning tool may return parameter sets that improve on the baseline only due to randomness. This parameter allows you to perform multiple solves for each parameter set, using different Seed values for each, in order to reduce the influence of randomness on the results.

Note: Only affects mixed integer programming (MIP) models

(tdefault = 2)

tuning (string): Parameter Tuning

Invokes the Gurobi parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Gurobi option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in MPS, REW, LP, RLP, and ILP format created by the WriteProb option. Gurobi will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after
the GAMS/Gurobi option file name without considering the problem provided by GAMS. The result of such a run is the updated GAMS/Gurobi option file with a tuned set of parameters. In case the option TuneResults is larger than 1, GAMS/Gurobi will create a sequence of GAMS/Gurobi option files. The solver and model status returned to GAMS will be NORMAL COMPLETION and NO SOLUTION. Tuning is incompatible with advanced features like FeasOpt of GAMS/Gurobi.

**usebasis (integer):** Use basis from GAMS

If usebasis is not specified, GAMS (option bratio) decides if the starting basis or a primal/dual solution is given to Gurobi. If usebasis is explicitly set in an option file then the basis or a primal/dual solution is passed to Gurobi independent of the GAMS option bratio. Please note, if Gurobi uses a starting basis presolve will be skipped.

(default = GAMS bratio)

- 0 No basis
- 1 Supply basis if basis is full otherwise provide primal dual solution
- 2 Supply basis iff basis is full
- 3 Supply primal dual solution

**varbranch (integer):** Controls the branch variable selection strategy

(default = -1)

- -1 Auto
- 0 Pseudo Reduced Cost Branching
- 1 Pseudo Shadow Price Branching
- 2 Maximum Infeasibility Branching
- 3 Strong Branching

**workerpassword (string):** Compute server password Pool of compute servers to use for distributed algorithms

When using a distributed algorithm (the distributed concurrent MIP solver or distributed tuning), this parameter allows you to specify the password for the workers listed in the WorkerPool parameter.

**workerpool (string):** Pool of compute servers to use for distributed algorithms

When using a distributed algorithm (the distributed concurrent solver, distributed MIP, or distributed tuning), this parameter allows you to specify a comma-separated list of workers. You can refer to workers using their names or their IP addresses. You should specify the access password, if there is one, in the WorkerPassword parameter.

To give an example, if you have two workers named server1.mydomain.com and server2.mydomain.com, with IP addresses 192.168.1.100 and 192.168.1.101, you could set the WorkerPool to “server1.mydomain.com,server2.mydomain.com” or “192.168.1.100,192.168.1.101”.

**writeparams (string):** Write Gurobi parameter file

**writeprob (string):** Save the problem instance

**zerohalfcuts (integer):** Controls zero-half cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

- -1 Auto
- 0 Off
- 1 Conservative
- 2 Aggressive

**zeroobjnodes (integer):** Number of nodes to explore in the zero objective heuristic

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a
feasible solution.
(default = 0)
1 Introduction

Knitro is a software package for finding local solutions of both continuous (i.e. smooth) optimization problems, with or without constraints, and discrete optimization problems with integer or binary variables. Even though Knitro has been designed for solving large-scale general problems, it is efficient for solving all of the following classes of optimization problems:

- unconstrained,
- bound constrained,
• equality constrained,
• systems of nonlinear equations,
• least squares problems,
• linear programming problems (LPs),
• quadratic programming problems (QPs),
• general (inequality) constrained problems,
• (convex) mixed integer nonlinear programs (MINLP) of moderate size.

The Knitro package provides the following features:

• Efficient and robust solution of small or large problems,
• Solvers for both continuous and discrete problems,
• Derivative-free, 1st derivative and 2nd derivative options,
• Both interior-point (barrier) and active-set optimizers,
• Both feasible and infeasible versions,
• Both iterative and direct approaches for computing steps,

The problems solved by Knitro have the form

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad c^L \leq c(x) \leq c^U \\
& \quad b^L \leq x \leq b^U,
\end{align*}
\]

(11.1a) (11.1b) (11.1c)

where the variables \( x \) can be continuous, binary, or integer. This allows many forms of constraints, including bounds on the variables. Knitro requires that the functions \( f(x) \) and \( c(x) \) be smooth functions.

Knitro implements both state-of-the-art interior-point and active-set methods for solving nonlinear optimization problems. In the interior method (also known as a barrier method), the nonlinear programming problem is replaced by a series of barrier sub-problems controlled by a barrier parameter \( \mu \). The algorithm uses trust regions and a merit function to promote convergence. The algorithm performs one or more minimization steps on each barrier problem, then decreases the barrier parameter, and repeats the process until the original problem (1) has been solved to the desired accuracy.

Knitro provides two procedures for computing the steps within the interior point approach. In the version known as Interior/CG each step is computed using a projected conjugate gradient iteration. This approach differs from most interior methods proposed in the literature in that it does not compute each step by solving a linear system involving the KKT (or primal-dual) matrix. Instead, it factors a projection matrix, and uses the conjugate gradient method, to approximately minimize a quadratic model of the barrier problem.

The second procedure for computing the steps, which we call Interior/Direct, always attempts to compute a new iterate by solving the primal-dual KKT matrix using direct linear algebra. In the case when this step cannot be guaranteed to be of good quality, or if negative curvature is detected, then the new iterate is computed by the Interior/CG procedure.

Knitro also implements an active-set sequential linear-quadratic programming (SLQP) algorithm which we call Active. This method is similar in nature to a sequential quadratic programming method but uses linear programming sub-problems to estimate the active-set at each iteration. This active-set code may be preferable when a good initial point can be provided, for example, when solving a sequence of related problems.

For problems with discrete variables, Knitro provides two variants of the branch and bound algorithm. The first is a standard implementation, while the second is specialized for convex, mixed-integer nonlinear problems.

We encourage the user to try all algorithmic options to determine which one is more suitable for the application at hand. For guidance on choosing the best algorithm see section Algorithm Options.
For a detailed description of the algorithm implemented in Interior/CG see [1] and for the global convergence theory see [2]. The method implemented in Interior/Direct is described in [8]. The Active algorithm is described in [5] and the global convergence theory for this algorithm is in [3]. An important component of Knitro is the HSL routine MA27 [6] which is used to solve the linear systems arising at every iteration of the algorithm. In addition, the Active algorithm in Knitro may make use of the COIN-OR Clp linear programming solver module. The version used in Knitro may be downloaded from http://www.ziena.com/clp.html.

2 Usage

Basic details of solver usage, including how to choose Knitro as the solver and how to use a solver-specific option file, are part of Chapter Basic Solver Usage.

As an NLP solver, Knitro can also be used to solve linear programs (LP), and both convex and nonconvex quadratic programs (QCP).

3 GAMS Options

The following GAMS options are used by the GAMS/Knitro link:

- Option ResLim = x;
  Sets the time limit in seconds. If this limit is exceeded the solver will terminate and pass on the current solution to GAMS. See also reslim in section GAMS options.

- Option SysOut = On;
  This option sends additional Knitro messages to the GAMS listing file. It is useful in case of a solver failure or to get algorithmic details. See also sysout in section GAMS options.

- ModelName.optCA = x;
  Absolute gap stop criterion for a discrete problem. The Knitro option mip_integral_gap_abs takes its default from this value. See also optca in section GAMS options.

- ModelName.optCR = x;
  Relative gap stop criterion for a discrete problem. The Knitro option mip_integral_gap_rel takes its default from this value. See also optcr in section GAMS options.

4 Summary of Knitro Options

The Knitro options file knitro.opt allows the user to easily set options controlling Knitro's behavior. Options are set by specifying a keyword and a corresponding value on a line in the knitro.opt file. Lines that begin with a # character are treated as comments and blank lines are ignored. For example, to set the maximum allowable number of iterations to 500, one could use the following options file:

4.1 Barrier options
### 4.2 General options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>controls which algorithm to use</td>
<td>0</td>
</tr>
<tr>
<td>delta</td>
<td>initial trust region radius scaling factor</td>
<td>1e-0</td>
</tr>
<tr>
<td>feastol</td>
<td>relative feasibility error tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>feastolabs</td>
<td>absolute feasibility error tolerance</td>
<td>0.0</td>
</tr>
<tr>
<td>gradopt</td>
<td>controls gradient computation</td>
<td>1</td>
</tr>
<tr>
<td>hessopt</td>
<td>controls Hessian computation</td>
<td>1</td>
</tr>
<tr>
<td>honorbnds</td>
<td>maintain feasibility of intermediate iterates w.r.t. variable bounds</td>
<td>0</td>
</tr>
<tr>
<td>lin_solver</td>
<td>controls which linear system solver to use</td>
<td>0</td>
</tr>
<tr>
<td>max_cgit</td>
<td>limit on inner CG iterations per minor iteration</td>
<td>0</td>
</tr>
<tr>
<td>maxit</td>
<td>major iteration limit</td>
<td>0</td>
</tr>
<tr>
<td>max_time_cpu</td>
<td>CPU time limit</td>
<td>1e8</td>
</tr>
<tr>
<td>max_time_real</td>
<td>real or wall-clock time limit</td>
<td>1e8</td>
</tr>
<tr>
<td>objrange</td>
<td>parameter used in unboundedness check</td>
<td>1e20</td>
</tr>
<tr>
<td>option_file</td>
<td>additional option file name - read only by KNITRO solver lib</td>
<td></td>
</tr>
<tr>
<td>opttol</td>
<td>relative optimality error tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>opttolabs</td>
<td>absolute optimality error tolerance</td>
<td>0.0</td>
</tr>
<tr>
<td>outlev</td>
<td>controls the level of output</td>
<td>2</td>
</tr>
<tr>
<td>output_time</td>
<td>print output on where time is used</td>
<td>0</td>
</tr>
<tr>
<td>pivot</td>
<td>initial pivot threshold used in the factorization routine</td>
<td>1e-8</td>
</tr>
<tr>
<td>presolve</td>
<td>control presolve level</td>
<td>1</td>
</tr>
<tr>
<td>reform</td>
<td>allow objective reformulation</td>
<td>1</td>
</tr>
<tr>
<td>scale</td>
<td>toggles problem scaling</td>
<td>1</td>
</tr>
<tr>
<td>soc</td>
<td>toggles the second order correction option</td>
<td>1</td>
</tr>
<tr>
<td>threads</td>
<td>default thread count</td>
<td>1</td>
</tr>
<tr>
<td>xtol</td>
<td>tolerance for termination on a small stepsize</td>
<td>1e-15</td>
</tr>
</tbody>
</table>

### 4.3 Multi-algorithm options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma_max_time_cpu</td>
<td>cumulative CPU time limit for multi-algorithm method</td>
<td>1e8</td>
</tr>
<tr>
<td>ma_max_time_real</td>
<td>cumulative real or wall-clock time limit for multi-algorithm method</td>
<td>1e8</td>
</tr>
</tbody>
</table>
4.4 MIP options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mip_branchrule</code></td>
<td>branching rule to use for MIP B&amp;B</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_gub_branch</code></td>
<td>toggles branching on generalized upper bounds</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_heuristic</code></td>
<td>MIP heuristic to use in searching for an initial integer feasible point</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_heuristic_maxit</code></td>
<td>maximum iterations to allow the MIP heuristic</td>
<td>100</td>
</tr>
<tr>
<td><code>mip_implications</code></td>
<td>toggles addition of constraints derived from logical implications</td>
<td>1</td>
</tr>
<tr>
<td><code>mip_integer_tol</code></td>
<td>integrality tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td><code>mip_integral_gap_abs</code></td>
<td>absolute stopping tolerance for MIP</td>
<td>1e-6</td>
</tr>
<tr>
<td><code>mip_integral_gap_rel</code></td>
<td>relative stopping tolerance for MIP</td>
<td>1e-6</td>
</tr>
<tr>
<td><code>mip_knapsack</code></td>
<td>knapsack cut generation control</td>
<td>1</td>
</tr>
<tr>
<td><code>mip_lpalg</code></td>
<td>algorithm to use for LP subproblems</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_maxnodes</code></td>
<td>maximum number of nodes to explore: 0=no limit</td>
<td>100000</td>
</tr>
<tr>
<td><code>mip_maxsolves</code></td>
<td>maximum number of subproblem solves: 0=no limit</td>
<td>200000</td>
</tr>
<tr>
<td><code>mip_maxtime_cpu</code></td>
<td>cumulative CPU time limit for MIP</td>
<td>1e8</td>
</tr>
<tr>
<td><code>mip_maxtime_real</code></td>
<td>cumulative real or wall-clock time limit for MIP</td>
<td>1e8</td>
</tr>
<tr>
<td><code>mip_method</code></td>
<td>specify MIP method to use</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_outinterval</code></td>
<td>node printing interval for MIP</td>
<td>10</td>
</tr>
<tr>
<td><code>mip_outlevel</code></td>
<td>how much MIP information to print</td>
<td>1</td>
</tr>
<tr>
<td><code>mip_pseudoinit</code></td>
<td>pseudocost initialization method control</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_rootalg</code></td>
<td>algorithm to use for the root node solve</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_rounding</code></td>
<td>MIP rounding rule to apply</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_selectrule</code></td>
<td>selection rule for the next node in the B&amp;B tree</td>
<td>0</td>
</tr>
<tr>
<td><code>mip_strong_candlim</code></td>
<td>max candidates to explore in strong branching</td>
<td>10</td>
</tr>
<tr>
<td><code>mip_strong_level</code></td>
<td>max levels on which to perform strong branching</td>
<td>10</td>
</tr>
<tr>
<td><code>mip_strong_maxit</code></td>
<td>max iterations to allow for strong branching</td>
<td>1000</td>
</tr>
<tr>
<td><code>mip_terminate</code></td>
<td>condition for terminating the MIP algorithm</td>
<td>0</td>
</tr>
</tbody>
</table>

4.5 Multi-start options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ms_deterministic</code></td>
<td>allow for deterministic parallel MS if <code>ms_terminate</code>=0</td>
<td>1</td>
</tr>
<tr>
<td><code>ms_enable</code></td>
<td>toggles multi-start method</td>
<td>0</td>
</tr>
<tr>
<td><code>ms_maxbndrange</code></td>
<td>maximum range to vary unbounded x when generating start points</td>
<td>1e3</td>
</tr>
<tr>
<td><code>ms_maxsolves</code></td>
<td>maximum number of start points to try during multi-start</td>
<td>0</td>
</tr>
<tr>
<td><code>ms_maxtime_cpu</code></td>
<td>cumulative CPU time limit for multi-start</td>
<td>1e8</td>
</tr>
<tr>
<td><code>ms_maxtime_real</code></td>
<td>cumulative real or wall-clock time limit for multi-start</td>
<td>1e8</td>
</tr>
<tr>
<td><code>ms_startptrange</code></td>
<td>maximum range to vary all x when generating start points</td>
<td>1e20</td>
</tr>
<tr>
<td><code>ms_terminate</code></td>
<td>termination condition for multi-start</td>
<td>0</td>
</tr>
</tbody>
</table>
5 Detailed Descriptions of Knitro Options

**algorithm (integer):** controls which algorithm to use

(default = 0)

0 automatic, based on problem characteristics
1 interior/direct
2 interior/CG
3 active-set CG method
4 active-set SQP method
5 multi-method, perhaps in parallel

**bar_feasible (integer):** specifies whether special emphasis is placed on getting and staying feasible

Indicates whether or not to use the feasible version of Knitro.

**NOTE:** This option can be used only with the Interior/CG and Interior/Direct algorithms, i.e. when \( \text{algorithm} = 2 \) or 3. See section Feasible version for more details.

Options 1 and 3 above activate the feasible version of KNITRO. Given an initial point which sufficiently satisfies all inequality constraints as defined by,

\[
cl + tol \leq c(x) \leq cu - tol
\]

(for \( cl \neq cu \)), the feasible version of Knitro ensures that all subsequent solution estimates strictly satisfy the inequality constraints. However, the iterates may not be feasible with respect to the equality constraints. The tolerance \( tol > 0 \) in (2) for determining when the feasible mode is active is determined by the double precision parameter \( \text{bar feasmodetol} \) described below. This tolerance (i.e. \( \text{bar feasmodetol} \)) must be strictly positive. That is, in order to enter feasible mode, the point given to Knitro must be strictly feasible with respect to the inequality constraints.

If the initial point is infeasible (or not sufficiently feasible according to (2)) with respect to the inequality constraints, then Knitro will run the infeasible version until a point is obtained which sufficiently satisfies all the inequality constraints. At this point it will switch to feasible mode.

(default = 0)

0 No special emphasis on feasibility
1 Iterates must satisfy inequality cons once they become sufficiently feasible
2 Special emphasis is placed on getting feasible before trying to optimize
3 Implement both options 1 and 2 above

**bar_feasmodetol (real):** tolerance for activation of bar_feasible mode

Specifies the tolerance in (2) by which the iterate must be feasible with respect to the inequality constraints before the feasible mode becomes active. This option is only relevant when \( \text{feasible} = 1 \).

(default = \( 1e-4 \))

**bar_initmu (real):** initial barrier parameter value

Specifies the initial value for the barrier parameter \( \mu \).

(default = \( 1e-1 \))

**bar_initpt (integer):** initial point strategy for barrier algorithms

Indicates whether an initial point strategy is used.

(default = 0)

0 let KNITRO choose the initial point strategy
1 shift the initial point to improve barrier performance
2 do not alter the initial point supplied by the user

**barrule (integer):** controls the barrier parameter update strategy

**NOTE:** Only strategies 0-2 are available for the Interior/CG algorithm. All strategies are available for the Interior/Direct algorithm. Strategies 4 and 5 are typically recommended for linear programs or convex quadratic programs.

(default = 0)

0 automatically choose the rule for updating the barrier parameter
1 monotonically decrease the barrier parameter
2 use an adaptive rule based on the complementarity gap to determine the value of the barrier parameter at every iteration
3 use a probing (affine-scaling) step
4 use a Mehrotra predictor-corrector type rule, with safeguards on the corrector step
5 use a Mehrotra predictor-corrector type rule, with no safeguards on the corrector step
6 minimize a quality function at each iteration

**barrefinement (integer):** toggles barrier solution refinement method

(default = 0)

**barswitchrule (integer):** controls switch to new feasibility-only phase

**NOTE:** The feasibility-only phase is new in Knitro 8.0. To get the behavior of older Knitro versions, choose strategy 1 (never switch).

(default = 0)

0 automatic
1 never switch to feasibility phase
2 allow switches to feasibility phase
3 more aggressive switches to feasibility phase

**delta (real):** initial trust region radius scaling factor

Specifies the initial trust region radius scaling factor used to determine the initial trust region size.

(default = 1e-0)

**feastol (real):** relative feasibility error tolerance

Specifies the final relative stopping tolerance for the feasibility error. Smaller values of feastol result in a higher degree of accuracy in the solution with respect to feasibility.

(default = 1e-6)

**feastolabs (real):** absolute feasibility error tolerance

Specifies the final absolute stopping tolerance for the feasibility error. Smaller values of feastolabs result in a higher degree of accuracy in the solution with respect to feasibility.

(default = 0.0)

**gradopt (integer):** controls gradient computation

Specifies how to compute the gradients of the objective and constraint functions.

(default = 1)

1 use exact gradients computed by GAMS
2 KNITRO computes gradients by forward finite differences
3 KNITRO computes gradients by central finite differences
**hessopt (integer):** controls Hessian computation

Specifies how to compute the (approximate) Hessian of the Lagrangian.

**NOTE:** In nearly all cases it is strongly recommended to use the exact Hessian option (option 1) or the exact Hessian-vector product option (option 5).

If exact Hessians (or exact Hessian-vector products) are not efficient to compute but exact gradients are provided and are not too expensive to compute, option 4 above is typically recommended. The finite-difference Hessian-vector option is comparable in terms of robustness to the exact Hessian option (assuming exact gradients are provided) and typically not too much slower in terms of time if gradient evaluations are not the dominant cost.

In the event that the exact Hessian (or Hessian-vector products) are too expensive to compute, multiple quasi-Newton options which internally approximate the Hessian matrix using first derivative information are provided. Options 2 and 3 are only recommended for small problems ($n < 1000$) since they require working with a dense Hessian approximation. Option 6 should be used in the large-scale case.

**NOTE:** Options hessopt= and hessopt=5 are not available when algorithm=1. See section Second derivative options for more detail on second derivative options.

(default = 1)

- 1 use exact Hessians computed by GAMS
- 2 use a dense quasi-Newton BFGS Hessian
- 3 use a dense quasi-Newton SR1 Hessian
- 4 compute Hessian-vector products using finite differences
- 5 use exact Hessian-vector products computed by GAMS
- 6 use a limited-memory quasi-Newton BFGS Hessian

**honorbnds (integer):** maintain feasibility of intermediate iterates w.r.t. variable bounds

Indicates whether or not to enforce satisfaction of the simple bounds (1c) throughout the optimization (see section Honor Bounds).

(default = 0)

- 0 does not enforce that the bounds on the variables are satisfied at intermediate iterates
- 1 enforces that the initial point and all subsequent solution estimates satisfy the bounds on the variables
- 2 enforces that the initial point satisfies the bounds on the variables

**lin solver (integer):** controls which linear system solver to use

Indicates which linear solver to use to solve linear systems arising in KNITRO algorithms.

(default = 0)

- 0 automatic: based on problem characteristics
- 1 internal: reserved for internal use, currently automatic
- 2 hybrid: linear solver used depends on the particular linear system to be solved
- 3 QR: use dense LAPACK QR routines, only suitable for small problems
- 4 MA27: use the HSL MA27 sparse symmetric indefinite solver
- 5 MA57: use the HSL MA57 sparse symmetric indefinite solver
- 6 PARDISO: use the Intel MKL PARDISO sparse symmetric indefinite solver

**max cg (integer):** limit on inner CG iterations per minor iteration

Specifies the maximum allowable number of inner conjugate gradient (CG) iterations per KNITRO minor iteration.

(default = 0)
0 upper bound determined automatically
n at most n CG iterations may be performed

maxcrossit (integer): limit on crossover iterations

Specifies the maximum number of crossover iterations before termination. If the value is positive, then KNITRO will crossover from the barrier to the Active Set algorithm near the solution. The Active Set algorithm will then perform at most n iterations to get a more exact solution. If the value is 0, no Active Set crossover occurs and the interior-point solution is the final result.

If Active Set crossover is unable to improve the approximate interior-point solution, then KNITRO will restore the interior-point solution. In some cases (especially on large-scale problems or difficult degenerate problems) the cost of the crossover procedure may be significant - for this reason, crossover is disabled by default. Enabling crossover generally provides a more accurate solution than Interior/Direct or Interior/CG.

(default = 0)

maxit (integer): major iteration limit

Specifies the maximum number of iterations before termination.

(default = 0)

0 automatically determines a value based on the problem size. Currently KNITRO 7.0 sets this value to 10000 for LPs/NLPs and 3000 for MIPs/MINLPs

n At most n iterations may be performed before terminating, where n > 0.

maxtime_cpu (real): CPU time limit

Specifies the CPU time limit, in seconds.

(default = 1e8)

maxtime_real (real): real or wall-clock time limit

Specifies the real or wall-clock time limit, in seconds.

(default = 1e8)

ma_maxtime_cpu (real): cumulative CPU time limit for multi-algorithm method

(default = 1e8)

ma_maxtime_real (real): cumulative real or wall-clock time limit for multi-algorithm method

(default = 1e8)

ma_terminate (integer): condition for terminating the multi-algorithm method

(default = 1)

0 terminate after all algorithms have completed
1 terminate at first local optimum
2 terminate at first feasible solution

mip_branchrule (integer): branching rule to use for MIP B&B

Branching rule to use for MIP B&B.

(default = 0)

0 automatic
1 use most fractional (most infeasible) branching
2 use pseudo-cost branching
3 use strong branching

mip_gub_branch (integer): toggles branching on generalized upper bounds

Toggles branching on generalized upper bounds.
mip_heuristic (integer): MIP heuristic to use in searching for an initial integer feasible point
Heuristic to use in searching for an initial integer feasible point.
(default = 0)

0 automatic
1 none
2 feasibility pump
3 heuristic based on MPEC formulation

mip_maxheuristic (integer): maximum iterations to allow the MIP heuristic
Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled.
(default = 100)

mip_implications (integer): toggles addition of constraints derived from logical implications
Toggles addition of constraints derived from logical implications
(default = 1)

mip_integer_tol (real): integrality tolerance
Specifies the integrality tolerance for discrete variables.
(default = 1e-8)

mip_integral_gap_abs (real): absolute stopping tolerance for MIP
The absolute integrality gap stop tolerance. If not set by the user, the GAMS optCA value is used.
(default = 1e-6)

mip_integral_gap_rel (real): relative stopping tolerance for MIP
The relative integrality gap stop tolerance. If not set by the user, the GAMS optCA value is used.
(default = 1e-6)

mip_knapsack (integer): knapsack cut generation control
(default = 1)

0 none
1 only for inequalities
2 for inequalities and equalities

mip_lpalg (integer): algorithm to use for LP subproblems
Specifies which algorithm to use for any LP subproblem solves that may occur in the B&B procedure. LP subproblems may arise if the problem has no nonlinear parts or if using mip_method=2.
(default = 0)

0 automatically try to choose the best algorithm based on the problem characteristics
1 use the Interior/Direct algorithm
2 use the Interior/CG algorithm
3 use the Active Set (simplex) algorithm

mip_maxnodes (integer): maximum number of nodes to explore: 0=no limit
Specifies the maximum number of nodes explored (0 means no limit)
(default = 100000)

mip_maxsolves (integer): maximum number of subproblem solves: 0=no limit
Specifies the maximum number of subproblem solves allowed (0 means no limit).
(default = 200000)

\texttt{mip\_maxtime\_cpu} (\textit{real}): cumulative CPU time limit for MIP

Specifies the cumulative CPU time limit, in seconds.
(default = 1e8)

\texttt{mip\_maxtime\_real} (\textit{real}): cumulative real or wall-clock time limit for MIP

Specifies the cumulative real or wall-clock time limit, in seconds.
(default = 1e8)

\texttt{mip\_method} (\textit{integer}): specify MIP method to use

Specifies which method to use.
(default = 0)

0 automatic
1 standard branch and bound method
2 hybrid Quesada-Grossman method (for convex, nonlinear problems only)

\texttt{mip\_outinterval} (\textit{integer}): node printing interval for MIP

Specifies node printing interval for \texttt{mip\_outlevel} when \texttt{mip\_outlevel} > 0.
(default = 10)

\texttt{mip\_outlevel} (\textit{integer}): how much MIP information to print

Specifies how much MIP information to print.
(default = 1)

0 do not print any MIP node information
1 print one line of output for every node

\texttt{mip\_pseudoinit} (\textit{integer}): pseudocost initialization method control

(default = 0)

0 automatic
1 use average value
2 use strong branching

\texttt{mip\_rootalg} (\textit{integer}): algorithm to use for the root node solve

Specifies which algorithm to use for the root node solve.
(default = 0)

0 automatic, based on problem characteristics
1 interior/direct
2 interior/CG
3 active-set method

\texttt{mip\_rounding} (\textit{integer}): MIP rounding rule to apply

Specifies the rounding rule to apply.
(default = 0)

0 automatic
1 do not round if a node is infeasible
2 round using a fast heuristic only
3 round and solve a subproblem if likely to succeed
4 always round and solve a subproblem

mip_selectrule (integer): selection rule for the next node in the B&B tree

Specifies the select rule for choosing the next node in the tree.
(default = 0)
0 automatic
1 search using a depth first procedure
2 select the node with the best relaxation bound
3 use depth first unless pruned, then best bound

mip_strong_candlim (integer): max candidates to explore for strong branching

Specifies the maximum number of candidates to explore for strong branching.
(default = 10)

mip_strong_level (integer): max levels on which to perform strong branching

Specifies the maximum number of tree levels on which to perform strong branching.
(default = 10)

mip_strong_maxit (integer): max iterations to allow for strong branching

Specifies the maximum number of iterations to allow for strong branching.
(default = 1000)

mip_terminate (integer): condition for terminating the MIP algorithm

Specifies conditions for terminating the MIP algorithm.
(default = 0)
0 terminate at optimum
1 terminate at first integer feasible point

ms_deterministic (integer): allow for deterministic parallel MS if ms_terminate=0

(default = 1)

ms_enable (integer): toggles multi-start method

Toggles multi-start method.
(default = 0)

ms_maxbndrange (real): maximum range to vary unbounded $x$ when generating start points

Maximum range to vary unbounded $x$ when generating start points.
(default = $1e3$)

ms_maxsolves (integer): maximum number of start points to try during multi-start

Specifies the maximum number of start points to try during multi-start.
(default = 0)
0 KNITRO sets the number based on problem size
$n$ try exactly $n > 0$ start points

ms_maxtime_cpu (real): cumulative CPU time limit for multi-start

Specifies the cumulative CPU time limit, in seconds.
(default = $1e8$)

ms_maxtime_real (real): cumulative real or wall-clock time limit for multi-start
Specifies the cumulative real or wall-clock time limit, in seconds.
(default = 1e8)

**ms_startp**range (real): maximum range to vary all $x$ when generating start points

Maximum range to vary all $x$ when generating start points.
(default = 1e20)

**ms_terminate** (integer): termination condition for multi-start

Specifies conditions for terminating the multi-start algorithm.
(default = 0)

0 terminate after ms_maxsolves
1 terminate at first local optimum (if before ms_maxsolves)
2 terminate at first feasible solution (if before ms_maxsolves)

**objrange** (real): parameter used in unboundedness check

Specifies the extreme limits of the objective function for purposes of determining unboundedness. If the magnitude of the objective function is greater than objrange and the iterate is feasible, then the problem is determined to be unbounded and Knitro proceeds no further.
(default = 1e20)

**option_file** (string): additional option file name - read only by KNITRO solver lib

**opttol** (real): relative optimality error tolerance

Specifies the final relative stopping tolerance for the KKT (optimality) error. Smaller values of opttol result in a higher degree of accuracy in the solution with respect to optimality.
(default = 1e-6)

**opttolabs** (real): absolute optimality error tolerance

Specifies the final absolute stopping tolerance for the KKT (optimality) error. Smaller values of opttolabs result in a higher degree of accuracy in the solution with respect to optimality.
(default = 0.0)

**outlev** (integer): controls the level of output

controls the level of output.
(default = 2)

0 printing of all output is suppressed
1 only summary information is printed
2 print basic information every 10 iterations
3 print basic information at each iteration
4 print basic information and the function count at each iteration
5 print all of the above, and the values of the solution vector $x$
6 print all of the above, and the values of the constraints $c$ and the Lagrange multipliers $\lambda$

**output_time** (integer): print output on where time is used

(default = 0)

**pivot** (real): initial pivot threshold used in the factorization routine

Specifies the initial pivot threshold used in the factorization routine. The value should be in the range $[0...0.5]$ with higher values resulting in more pivoting (more stable factorizations). Values less than 0 will be set to 0 and values larger than 0.5 will be set to 0.5. If pivot is non-positive initially no pivoting will be performed. Smaller values may improve the speed of the code but higher values are recommended for more stability (for example, if
the problem appears to be very ill-conditioned).
(default = 1e-8)

**presolve (integer): control presolve level**

Controls the presolve level.
(default = 1)
- 0 no presolve
- 1 basic presolve
- 2 advanced presolve

**reform (integer): allow objective reformulation**

(default = 1)

**scale (integer): toggles problem scaling**

Performs a scaling of the objective and constraint functions based on their values at the initial point. If scaling is performed, all internal computations, including the stopping tests, are based on the scaled values.
(default = 1)
- 0 No scaling is performed.
- 1 The objective function and constraints may be scaled.

**soc (integer): toggles the second order correction option**

Specifies whether or not to try second order corrections (SOC). A second order correction may be beneficial for problems with highly nonlinear constraints.
(default = 1)
- 0 No second-order correction steps are attempted.
- 1 Second-order correction steps may be attempted.
- 2 Second-order correction steps are always attempted if the original step is rejected and there are nonlinear constraints.

**threads (integer): default thread count**

Controls the number of threads to use. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.
(default = 1)

**xtol (real): tolerance for termination on a small stepsize**

Specifies when to terminate the optimization based on stepsize. The optimization will terminate when the relative change in the solution estimate is less than xtol. If using an interior-point algorithm and the barrier parameter is still large, Knitro will first try decreasing the barrier parameter before terminating.
(default = 1e-15)
6 Knitro Termination Test and Optimality

6.1 Continuous problems

The first-order conditions for identifying a locally optimal solution of the problem (1) are:

$$
\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) + \sum_{i=1}^{m} \lambda_i^c \nabla c_i(x) + \sum_{j=1}^{n} \lambda_j^b \nabla b_j(x) = 0
$$

$$
\lambda_i^c \min[(c_i(x) - c_i^l), (c_i^u - c_i(x))] = 0, \quad i = 1 \ldots m
$$

$$
\lambda_j^b \min[(x_j - b_j^l), (b_j^u - x_j)] = 0, \quad j = 1 \ldots n
$$

$$
c_i^l \leq c_i(x) \leq c_i^u, \quad i = 1 \ldots m
$$

$$
b_j^l \leq x_j \leq b_j^u, \quad j = 1 \ldots n
$$

(3)

where $\mathcal{I}$ and $\mathcal{B}$ represent the sets of indices corresponding to the general inequality constraints and non-fixed variable bound constraints respectively, $\lambda_i^c$ is the Lagrange multiplier corresponding to constraint $c_i(x)$, and $\lambda_j^b$ is the Lagrange multiplier corresponding to the simple bounds on $x_j$. There is exactly one Lagrange multiplier for each constraint and variable. The Lagrange multiplier may be restricted to take on a particular sign depending on the corresponding constraint or variable bounds.

In Knitro we define the feasibility error ($\text{FeasErr}$) at a point $x^k$ to be the maximum violation of the constraints of (1), i.e.,

$$
\text{FeasErr} = \max_{i=1 \ldots m, j=1 \ldots n} (0, (c_i^l - c_i(x^k)), (c_i(x^k) - c_i^u), (b_j^l - x_j^k), (x_j^k - b_j^u)),
$$

while the optimality error ($\text{OptErr}$) is defined as the maximum violation of the first three conditions of (3)

$$
\text{OptErr} = \max_{i=1 \ldots m, j=1 \ldots n} \left( \|\nabla_x \mathcal{L}(x, \lambda^k)\|_\infty, \lambda_i^c \min[(c_i(x) - c_i^l), (c_i^u - c_i(x))], \lambda_j^b \min[(x_j - b_j^l), (b_j^u - x_j)] \right).
$$

The remaining conditions on the sign of the multipliers are enforced explicitly throughout the optimization. In order to take into account problem scaling in the termination test, the following scaling factors are defined. In order to take into account problem scaling in the termination test, the following scaling factors are defined

$$
\tau_1 = \max(1, (c_i^l - c_i(x^0)), (c_i(x^0) - c_i^u), (b_j^l - x_j^0), (x_j^0 - b_j^u)),
$$

$$
\tau_2 = \begin{cases} 
\max(1, \|\nabla f(x^0)\|_\infty), & \text{for constrained problems}, \\
\max(1, \min(\|f(x^0)\|, \|\nabla f(x^0)\|_\infty)), & \text{for unconstrained problems}, 
\end{cases}
$$

where $x^0$ represents the initial point. The special treatment for unconstrained problems is necessary, as for these problems, $\|\nabla f(x^k)\|_\infty \to 0$ as a solution is approached, thus $\max(1, \|\nabla f(x^0)\|_\infty)$ would not be effective.

Knitro stops and declares a locally optimal solution found if the following stopping conditions are satisfied:

$$
\text{FeasErr} \leq \max(\tau_1 \ast \text{feastol}, \text{feastolabs})
$$

$$
\text{OptErr} \leq \max(\tau_2 \ast \text{opttol}, \text{opttolabs})
$$

(4)

(5)

where $\text{feastol}$, $\text{opttol}$, $\text{feastolabs}$ and $\text{opttolabs}$ are user-defined options (see section Usage).

This stopping test is designed to give the user much flexibility in deciding when the solution returned by Knitro is accurate enough. One can use a purely scaled stopping test (which is the recommended and default option) by setting $\text{feastolabs}$
and opttolabs equal to 0.0e0. Likewise, an absolute stopping test can be enforced by setting feastol and opttol equal to 0.0e0.

**Unbounded problems**

Since by default Knitro uses a relative/scaled stopping test it is possible for the optimality conditions to be satisfied for an unbounded problem. For example, if \( t_2 \to \infty \) while the optimality error OptErr stays bounded, condition (5) will eventually be satisfied for some opttol>0. If you suspect that your problem may be unbounded, using an absolute stopping test will allow Knitro to detect this.

### 6.2 Discrete problems

Algorithms for solving versions of (1) where one or more of the variables are restricted to take on only discrete values, proceed by solving a sequence of continuous relaxations, where the discrete variables are relaxed such that they can take on any continuous value. The global solutions \( f(x_R) \) of these relaxed problems provide a lower bound on the optimal objective value for problem (1) (upper bound if maximizing). If a feasible point is found for problem (1) that satisfies the discrete restrictions on the variables, then this provides an upper bound on the optimal objective value of problem (1) (lower bound if maximizing). We will refer to these feasible points as incumbent points and denote the objective value at an incumbent point by \( f(x_I) \). Assuming all the continuous subproblems have been solved to global optimality (if the problem is convex, all local solutions are global solutions), an optimal solution of problem (1) is verified when the lower bound and upper bound are equal.

Knitro declares optimality for a discrete problem when the gap between the best (i.e., largest) lower bound \( f(x_R) \) and the best (i.e., smallest) upper bound \( f(x_I) \) is less than a threshold determined by the user options mip_integral_gap_abs and mip_integral_gap_rel. Specifically, Knitro declares optimality when either

\[
f(x_I) - f(x_R) \leq \text{mip_integral_gap_abs}
\]

or

\[
f(x_I) - f(x_R) \leq \text{mip_integral_gap_rel} \cdot \max(1,|f(x_I)|),
\]

where \( \text{mip_integral_gap_abs} \) and \( \text{mip_integral_gap_rel} \) are typically small positive numbers. Since these termination conditions assume that the continuous subproblems are solved to global optimality and Knitro only finds local solutions of nonconvex, continuous optimization problems, they are only reliable when solving convex, mixed integer problems. The integrality gap \( f(x_I) - f(x_R) \) should be non-negative although it may become slightly negative from roundoff error, or if the continuous subproblems are not solved to sufficient accuracy. If the integrality gap becomes largely negative, this may be an indication that the model is nonconvex, in which case Knitro may not converge to the optimal solution, and will be unable to verify optimality (even if it claims otherwise).

Note that the default values for \( \text{mip_integral_gap_abs} \) and \( \text{mip_integral_gap_rel} \) are taken from the GAMS options optCA and optCR, but an explicit setting of \( \text{mip_integral_gap_abs} \) or \( \text{mip_integral_gap_rel} \) will override those.

### 7 Knitro Output

If outlev=0 then all printing of output is suppressed. The description below assumes the default output level (outlev=2) except where indicated:

**Nondefault Options:**

This output lists all user options (see section Usage) which are different from their default values. If nothing is listed in this section then all user options are set to their default values.

**Problem Characteristics:**

The output begins with a description of the problem characteristics.

**Iteration Information - Continuous Problems:**

An iteration, in the context of Knitro, is defined as a step which generates a new solution estimate (i.e., a successful step). The columns of the iteration log are as follows:
• Iter  Iteration number.
• fCount  The cumulative number of function evaluations, only included if (outlev>3)
• Objective  Gives the value of the objective function at the current iterate.
• FeasErr  Gives a measure of the feasibility violation at the current iterate.
• OptErr  Gives a measure of the violation of the Karush-Kuhn-Tucker (KKT) (first-order) optimality conditions (not including feasibility) at the current iterate.
• ||Step||  The 2-norm length of the step (i.e., the distance between the new iterate and the previous iterate).
• CG its  The number of Projected Conjugate Gradient (CG) iterations required to compute the step.

If outlev=2, information is printed every 10 major iterations. If outlev=3 information is printed at each major iteration. If outlev>4 addition information is included in the log.

Iteration Information - Discrete Problems:
By default, the GAMS/Knitro link prints a log line at every 10’th node. This frequency can be changed via the mip_outinterval option. To turn off the node log completely, set the mip_outlevel option to 0. The columns of the iteration log for discrete models are as follows:

• Node  The node number. If an integer feasible point was found at a given node, it is marked with a *
• Left  The current number of active nodes left in the branch and bound tree.
• Inf  The current number of active nodes left in the branch and bound tree.
• Objective  Gives the value of the objective function at the solution of the relaxed subproblem solved at the current node. If the subproblem was infeasible or failed, this is indicated. Additional symbols may be printed at some nodes if the node was pruned (pr), integer feasible (f), or an integer feasible point was found through rounding (r).
• Best relaxatn  The value of the current best relaxation (lower bound on the solution if minimizing).
• Best incumbent  The value of the current best integer feasible point (upper bound on the solution if minimizing).

Termination Message:  At the end of the run a termination message is printed indicating whether or not the optimal solution was found and if not, why the solver terminated. Below is a list of some possible termination messages.

• EXIT: Locally optimal solution found.
  Knitro found a locally optimal point which satisfies the stopping criterion (see section Knitro Termination Test and Optimality) for more detail on how this is defined). If the problem is convex (for example, a linear program), then this point corresponds to a globally optimal solution.
• EXIT: Iteration limit reached.
  The iteration limit was reached before being able to satisfy the required stopping criteria.
• EXIT: Convergence to an infeasible point. Problem appears to be locally infeasible.
  The algorithm has converged to an infeasible point from which it cannot further decrease the infeasibility measure. This happens when the problem is infeasible, but may also occur on occasion for feasible problems with nonlinear constraints or badly scaled problems. It is recommended to try various initial points. If this occurs for a variety of initial points, it is likely the problem is infeasible.
• EXIT: Problem appears to be unbounded.
  The objective function appears to be decreasing without bound, while satisfying the constraints.
• EXIT: Current point cannot be improved.
No more progress can be made. If the current point is feasible it is likely it may be optimal, however the stopping tests cannot be satisfied perhaps because of degeneracy, ill-conditioning or bad scaling).

- **EXIT:** Current point cannot be improved. Point appears to be optimal, but desired accuracy could not be achieved.
  
  No more progress can be made, but the stopping tests are close to being satisfied (within a factor of 100) and so the current approximate solution is believed to be optimal.

- **EXIT:** Time limit reached.
  
  The time limit was reached before being able to satisfy the required stopping criteria.

- **EXIT:** Evaluation error.
  
  This termination value indicates that an evaluation error occurred (e.g., divide by 0, taking the square root of a negative number), preventing the optimization from continuing.

- **EXIT:** Not enough memory available to solve problem.
  
  This termination value indicates that there was not enough memory available to solve the problem.

**Final Statistics:**

Following the termination message some final statistics on the run are printed. Both relative and absolute error values are printed.

**Solution Vector/Constraints:**

If `outlev`=5, the values of the solution vector are printed after the final statistics. If `outlev`=6, the final constraint values are also printed before the solution vector and the values of the Lagrange multipliers (or dual variables) are printed next to their corresponding constraint or bound.

## 8 Algorithm Options

### 8.1 Automatic

By default, Knitro will automatically try to choose the best optimizer for the given problem based on the problem characteristics.

### 8.2 Interior/Direct

If the Hessian of the Lagrangian is ill-conditioned or the problem does not have a large-dense Hessian, it may be advisable to compute a step by directly factoring the KKT (primal-dual) matrix rather than using an iterative approach to solve this system. Knitro offers the Interior/Direct optimizer which allows the algorithm to take direct steps by setting `algorithm`=1. This option will try to take a direct step at each iteration and will only fall back on the iterative step if the direct step is suspected to be of poor quality, or if negative curvature is detected.

Using the Interior/Direct optimizer may result in substantial improvements over Interior/CG when the problem is ill-conditioned (as evidenced by Interior/CG taking a large number of Conjugate Gradient iterations). We encourage the user to try both options as it is difficult to predict in advance which one will be more effective on a given problem. In each case, also experiment with the `bar_murule` option, as it is difficult to predict which update rule will work best.

**NOTE:** Since the Interior/Direct algorithm in Knitro requires the explicit storage of a Hessian matrix, this version can only be used with Hessian options, `hessopt`=1, 2, 3 or 6. It may not be used with Hessian options, `hessopt`=4 or 5, which only provide Hessian-vector products. Both the Interior/Direct and Interior/CG methods can be used with the `bar_feasible` option.
8.3 Interior/CG

Since Knitro was designed with the idea of solving large problems, the Interior/CG optimizer in Knitro offers an iterative Conjugate Gradient approach to compute the step at each iteration. This approach has proven to be efficient in most cases and allows Knitro to handle problems with large, dense Hessians, since it does not require factorization of the Hessian matrix. The Interior/CG algorithm can be chosen by setting algorithm=2. It can use any of the Hessian options as well as the bar_feasible option.

8.4 Active Set

Knitro includes an active-set Sequential Linear-Quadratic Programing (SLQP) optimizer. This optimizer is particular advantageous when "warm starting" (i.e., when the user can provide a good initial solution estimate, for example, when solving a sequence of closely related problems). This algorithm is also the preferred algorithm for detecting infeasible problems quickly. The Active Set algorithm can be chosen by setting algorithm=3. It can use any of the Hessian options.

9 Other Knitro special features

This section describes in more detail some of the most important features of Knitro and provides some guidance on which features to use so that Knitro runs most efficiently for the problem at hand.

9.1 Second derivative options

The default version of Knitro assumes that exact second derivatives of the objective function and constraint functions can be computed. If this is possible and the cost of computing the second derivatives is not overly expensive, it is highly recommended to use exact second derivatives. However, Knitro also offers other options which are described in detail below.

(Dense) Quasi-Newton BFGS
The quasi-Newton BFGS option uses gradient information to compute a symmetric, positive-definite approximation to the Hessian matrix. Typically this method requires more iterations to converge than the exact Hessian version. However, since it is only computing gradients rather than Hessians, this approach may be more efficient in many cases. This option stores a dense quasi-Newton Hessian approximation so it is only recommended for small to medium problems (n < 1000). The quasi-Newton BFGS option can be chosen by setting options value hessopt=2.

(Dense) Quasi-Newton SR1
As with the BFGS approach, the quasi-Newton SR1 approach builds an approximate Hessian using gradient information. However, unlike the BFGS approximation, the SR1 Hessian approximation is not restricted to be positive-definite. Therefore the quasi-Newton SR1 approximation may be a better approach, compared to the BFGS method, if there is a lot of negative curvature in the problem since it may be able to maintain a better approximation to the true Hessian in this case. The quasi-Newton SR1 approximation maintains a dense Hessian approximation and so is only recommended for small to medium problems (n < 1000). The quasi-Newton SR1 option can be chosen by setting options value hessopt=3.

Finite-difference Hessian-vector product option
If the problem is large and gradient evaluations are not the dominate cost, then Knitro can internally compute Hessian-vector products using finite-differences. Each Hessian-vector product in this case requires one additional gradient evaluation. This option can be chosen by setting options value hessopt=4. This option is generally only recommended if the exact gradients are provided.

NOTE: This option may not be used when algorithm=1.

Exact Hessian-vector products
In some cases the problem which the user wishes to solve may have a large, dense Hessian which makes it impractical to store or work with the Hessian directly. The performance of this option should be nearly identical to the exact Hessian option but requires much less storage. This option can be chosen by setting options value hessopt=5.

NOTE: This option may not be used when algorithm=1.
**Limited-memory Quasi-Newton BFGS**

The limited-memory quasi-Newton BFGS option is similar to the dense quasi-Newton BFGS option described above. However, it is better suited for large-scale problems since, instead of storing a dense Hessian approximation, it only stores a limited number of gradient vectors used to approximate the Hessian. In general it requires more iterations to converge than the dense quasi-Newton BFGS approach but will be much more efficient on large-scale problems. This option can be chosen by setting options value hessopt=6.

**9.2 Feasible version**

Knitro offers the user the option of forcing intermediate iterates to stay feasible with respect to the *inequality* constraints (it does not enforce feasibility with respect to the *equality* constraints however). Given an initial point which is sufficiently feasible with respect to all inequality constraints and selecting bar_feasible = 1, forces all the iterates to strictly satisfy the inequality constraints throughout the solution process. For the feasible mode to become active the iterate \( x \) must satisfy

\[
cl + tol \leq c(x) \leq cu - tol
\]

for all inequality constraints (i.e., for \( cl \neq cu \)). The tolerance \( tol > 0 \) by which an iterate must be strictly feasible for entering the feasible mode is determined by the parameter bar_feasmodetol which is 1.0e-4 by default. If the initial point does not satisfy (21) then the default infeasible version of Knitro will run until it obtains a point which is sufficiently feasible with respect to all the inequality constraints. At this point it will switch to the feasible version of Knitro and all subsequent iterates will be forced to satisfy the inequality constraints.

For a detailed description of the feasible version of Knitro see [4].

**NOTE:** This option may only be used when algorithm=2.

**9.3 Honor Bounds**

By default Knitro does not enforce that the simple bounds on the variables (1c) are satisfied throughout the optimization process. Rather, satisfaction of these bounds is only enforced at the solution. In some applications, however, the user may want to enforce that the initial point and all intermediate iterates satisfy the bounds \( bl \leq x \leq bu \). This can be enforced by setting honorbnds=1.

**9.4 Crossover**

Interior-point (or barrier) methods are a powerful tool for solving large-scale optimization problems. However, one drawback of these methods is that they do not always provide a clear picture of which constraints are active at the solution. In general they return a less exact solution and less exact sensitivity information. For this reason, Knitro offers a crossover feature in which the interior-point method switches to the Active Set method at the interior-point solution estimate, in order to “clean up” the solution and provide more exact sensitivity and active set information. The crossover procedure is controlled by the maxcrossit option. If this option is greater than 0, then Knitro will attempt to perform maxcrossit Active Set crossover iterations after the interior-point method has finished, to see if it can provide a more exact solution. This can be viewed as a form of post-processing. If maxcrossit is not positive, then no crossover iterations are attempted.

The crossover procedure will not always succeed in obtaining a more exact solution compared with the interior-point solution. If crossover is unable to improve the solution within maxcrossit crossover iterations, then it will restore the interior-point solution estimate and terminate. By default, Knitro will then print a message indicating that it was unable to improve the solution within the iterations allowed. In this case, you may want to increase the value of maxcrossit and try again. If Knitro determines that the crossover procedure will not succeed, no matter how many iterations are tried, then a message of the form Crossover mode unable to improve solution. will be printed.

The extra cost of performing crossover is problem dependent. In most small or medium scale problems, the crossover cost is a small fraction of the total solve cost. In these cases it may be worth using the crossover procedure to obtain a more exact solution. On some large scale or difficult degenerate problems, however, the cost of performing crossover may be significant. It is recommended to experiment with this option to see whether improvement in the exactness of the solution is worth the additional cost.
9.5 Solving Systems of Nonlinear Equations

Knitro is quite effective at solving systems of nonlinear equations. To solve a square system of nonlinear equations using Knitro one should specify the nonlinear equations as equality constraints (i.e., constraints with \( cl = cu \)), and specify the objective function (1a) as zero (i.e., \( f(x) = 0 \)).

9.6 Solving Least Squares Problems

There are two ways of using Knitro for solving problems in which the objective function is a sum of squares of the form

\[
f(x) = \frac{1}{2} \sum_{j=1}^{q} r_j(x)^2.
\]

If the value of the objective function at the solution is not close to zero (the large residual case), the least squares structure of \( f \) can be ignored and the problem can be solved as any other optimization problem. Any of the Knitro options can be used.

On the other hand, if the optimal objective function value is expected to be small (small residual case) then Knitro can implement the Gauss-Newton or Levenberg-Marquardt methods which only require first derivatives of the residual functions, \( r_j(x) \), and yet converge rapidly. To do so, the user need only define the Hessian of \( f \) to be

\[
\nabla^2 f(x) = J(x)^T J(x),
\]

where

\[
J(x) = \left[ \frac{\partial r_j}{\partial x_i} \right]_{j=1}^{q}, \quad i = 1, 2, \ldots, n
\]

The actual Hessian is given by

\[
\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^{q} r_j(x) \nabla^2 r_j(x);
\]

the Gauss-Newton and Levenberg-Marquardt approaches consist of ignoring the last term in the Hessian.

Knitro will behave like a Gauss-Newton method by setting \( \text{algorithm}=1 \), and will be very similar to the classical Levenberg-Marquardt method when \( \text{algorithm}=2 \). For a discussion of these methods see, for example, [7].

Bibliography


1 Introduction

1.1 The LGO Solver Suite

The Lipschitz-Continuous Global Optimizer \(^1\) (LGO) serves for the analysis and global solution of general nonlinear programming (NLP) models. The LGO solver system has been developed and gradually extended for more than a decade and it now incorporates a suite of robust and efficient global and local nonlinear solvers. It can also handle small LP models.

GAMS/LGO can be used in several search modes, providing a robust, effective, and flexible solver suite approach to a broad range of nonlinear models. The solver suite approach increases the reliability of the overall solution process. GAMS/LGO integrates the following global scope algorithms:

- Branch-and-bound (adaptive partition and sampling) based global search (BB)
- Adaptive global random search (GARS)
- Adaptive multistart global random search (MS)

LGO also includes the following local solver strategies:

- Heuristic global scope scatter search method (HSS)
- Bound-constrained local search, based on the use of an exact penalty function (EPM)
- Constrained local search, based on sequential model linearization (SLP)
- Constrained local search, based on a generalized reduced gradient approach (GRG).

\(^1\) Also see http://www.dal.ca/~jdpinter/l_s_d.html
The overall solution approach followed by GAMS/LGO is based on the seamless combination of the global and local search strategies. This allows for a broad range of operations. In particular, a solver suite approach supports the flexible usage of the component solvers: one can execute fully automatic (global and/or local search based) optimization, and can design customized interactive runs.

GAMS/LGO does not rely on any sub-solvers, and it does not require any structural information about the model. It is particularly suited to solve even ‘black box’ (closed, confidential), or other complex models, in which the available analytical information may be limited. GAMS/LGO needs only computable function values (without a need for higher order analytical information). GAMS/LGO can even solve models having constraints involving continuous, but non-differentiable functions. Thus, within GAMS, LGO is well suited to solve DNLP models.

GAMS/LGO can also be used in conjunction with other GAMS solvers. For instance, the local solver CONOPT can be used after LGO is finished to verify the solution and/or to provide additional information such as marginal values. To call CONOPT, the user can specify the LGO solver option `callConopt`. See the LGO Options section for details.

The LGO solver suite has been successfully applied to complex, large-scale models both in educational/research and commercial contexts for over a decade. Possible application areas include advanced engineering design, econometrics and finance, medical research and biotechnology, chemical and process industries, and scientific modeling. Tractable model sizes depend only on the available hardware, although LGO has a 3000 variable, 2000 constraint size limit.

For more information, we refer to

- GAMS/LGO Nonlinear Solver Suite: Key Features, Usage, and Numerical Performance
- Nonlinear Optimization with GAMS/LGO (2006)

1.2 Running GAMS/LGO

GAMS/LGO is capable of solving the following model types: LP, RMIP, NLP, and DNLP. If LGO is not specified as the default solver for these models, it can be invoked by issuing the following command before the solve statement:

```gams
option (modeltype) = lgo;
```

where `modeltype` stands for LP, RMIP, NLP, or DNLP.

2 LGO Options

GAMS/LGO works like other GAMS solvers, and many options can be set directly within the GAMS model. The most relevant GAMS options are `reslim`, `iterlim`, and `optfile`. A description of all available GAMS options can be found in the Chapter “Using Solver Specific Options”. See the GAMS Solver Manuals.

If you specify "<modelname>.optfile = 1;" before the SOLVE statement in your GAMS model, GAMS/LGO will then look for and read an option file with the name lgo.opt (see "Using Solver Specific Options" for general use of solver option files). The syntax for the LGO option file is

```gams
optname = value
```

with one option on each line. For example, one can write

```gams
opmode = 1
```

This specifies LGO to use global branch and bound search and the built-in local search methods.

2.1 General LGO options
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc_tr</td>
<td>Global search termination - acceptability threshold Global search termination criterion parameter (acceptability threshold). The global search phase (BB, GARS, or MS) ends, if an overall merit function value is found in the global search phase that is not greater than acc_tr.</td>
<td>-1e10</td>
</tr>
<tr>
<td>con_tol</td>
<td>Maximal constraint violation tolerance in local search</td>
<td>1e-6</td>
</tr>
<tr>
<td>fct_trg</td>
<td>Partial stopping criterion in second local search phase</td>
<td>-1e10</td>
</tr>
<tr>
<td>fi_tol</td>
<td>Local search (merit function improvement) tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>g_maxfct</td>
<td>Maximum number of function evaluations in global search Maximum number of merit function evaluations before termination of global search phase (BB, GARS, or MS). In the default setting, n is the number of variables and m is the number of constraints. The difficulty of global optimization models varies greatly: for difficult models, g_maxfct can be increased as deemed necessary.</td>
<td>-1</td>
</tr>
<tr>
<td>irngs</td>
<td>Random number seed</td>
<td>0</td>
</tr>
<tr>
<td>kt_tol</td>
<td>Kuhn-Tucker local optimality condition violation tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>max_nosuc</td>
<td>Maximum number of function evaluations without improvement Maximum number of merit function evaluations in global search phase (BB, GARS, or MS) where no improvement is made. Algorithm phase terminates upon reaching this limit. The default of -1 uses 100(nvars+ncons), where nvars is the number of variables and ncons the number of constraints.</td>
<td>-1</td>
</tr>
<tr>
<td>opmode</td>
<td>Operational mode 0 LS Local search from the given nominal solution without a preceding local search (LS) 1 BB + LS Global branch-and-bound search and local search (BB+LS) 2 RS+LS Global adaptive random search and local search (GARS+LS) 3 MS+LS Global multistart random search and local search (MS+LS)</td>
<td>3</td>
</tr>
<tr>
<td>penmult</td>
<td>Constraint penalty multiplier. Global merit function is defined as objective + the constraint violations weighted by penmult.</td>
<td>100</td>
</tr>
<tr>
<td>tlimit</td>
<td>Time limit in seconds. This is equivalent to the GAMS option reslim. If specified, this overrides the GAMS reslim option.</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Gams system interface only

#### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad_obj</td>
<td>Default value for objective function, if evaluation errors occur</td>
<td>1e8</td>
</tr>
<tr>
<td>CallConopt</td>
<td>call CONOPT to get duals for final point Number of seconds given for cleanup phase using CONOPT. CONOPT terminates after at most CallConopt seconds. The cleanup phase determines duals for final solution point.</td>
<td>5</td>
</tr>
<tr>
<td>debug</td>
<td>Debug option. Prints out complete LGO status report to listing file.</td>
<td>0</td>
</tr>
<tr>
<td>Include</td>
<td>Start reading from a new file</td>
<td></td>
</tr>
<tr>
<td>log_err</td>
<td>Log first log_err errors Error reported (if applicable) every log_err iterations.</td>
<td>10</td>
</tr>
<tr>
<td>log_iter</td>
<td>Iteration log time interval. Log output occurs every log_iter iterations.</td>
<td>10</td>
</tr>
<tr>
<td>log_time</td>
<td>Iteration log time interval in seconds. Log output occurs every log_time seconds.</td>
<td>0.5</td>
</tr>
<tr>
<td>var_lo</td>
<td>Smallest (default) lower bound, unless set by user.</td>
<td>-1e6</td>
</tr>
</tbody>
</table>
3 The GAMS/LGO Log File

The GAMS/LGO log file gives much useful information about the current solver progress and its individual phases. For illustration, we use the nonconvex model mhw4d.gms from the GAMS model library:

```
$Title Nonlinear Test Problem (MHW4D,SEQ=84)

$Ontext
Another popular testproblem for NLP codes.

$Offtext

Variables m, x1, x2, x3, x4, x5;
Equations funct, eq1, eq2, eq3;

funct.. m =e= sqr(x1-1) + sqr(x1-x2) + power(x2-x3,3) + power(x3-x4,4) + power(x4-x5,4);
eq1.. x1 + sqr(x2) + power(x3,3) =e= 3*sqrt(2) + 2;
eq2.. x2 - sqr(x3) + x4 =e= 2*sqrt(2) - 2;
eq3.. x1*x5 =e= 2;

Model wright / all /;

x1.l = -1; x2.l = 2; x3.l = 1; x4.l = -2; x5.l = -2;
Solve wright using nlp minimizing m;
```

Note that the solution given by LGO (shown on the next page) corresponds to the global minimum. For comparison, note that local scope nonlinear solvers will not find the global solution, unless started from a suitable neighbourhood (i.e., the model-and solver-specific region of attraction) of that solution.

In this example we use an option file to print out log information every $500$ iterations, regardless of the elapsed time. Note that we set the log time option to 0 to ignore the log time interval.

```
LGO 1.0 May 15, 2003 LNX.LG.NA 21.0 001.000.000.LXI Lib001-030502

LGO Lipschitz Global Optimization
(C) Pinter Consulting Services, Inc.
129 Glenforest Drive, Halifax, NS, Canada B3M 1J2
E-mail : jdpinter@hfx.eastlink.ca
Website: www.dal.ca/~jdpinter

--- Using option file C:/GAMSPROJECTS/LGDOC/LGO.OPT
> log_iter 500
> log_time 0

3 defined, 0 fixed, 0 free
6 +/- INF bound(s) have been reset
1 LGO equations and 3 LGO variables
```

The first part prints out information about the model size after presolve. In this particular problem, the original model had 4 rows, 6 columns, and 14 non-zeroes, of which 3 were defined constraints, meaning that they could be eliminated via
GAMS/LGO presolve techniques. Note that none of these were fixed or free constraints. Furthermore, LGO presolve reduced the model size further to 1 row (LGO equations) and 3 columns (LGO variables).

The main log gives information for every $n$ iterations about current progress. The main fields are given in the table below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter</td>
<td>Current iteration.</td>
</tr>
<tr>
<td>Objective</td>
<td>Current objective function value.</td>
</tr>
<tr>
<td>SumInf</td>
<td>Sum of constraint infeasibilities.</td>
</tr>
<tr>
<td>MaxInf</td>
<td>Maximum constraint infeasibility.</td>
</tr>
<tr>
<td>Seconds</td>
<td>Current elapsed time in seconds.</td>
</tr>
<tr>
<td>Errors</td>
<td>Number of errors and type. Type can either be</td>
</tr>
<tr>
<td></td>
<td>D/E: Evaluation error</td>
</tr>
<tr>
<td></td>
<td>B: Bound violation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iter</th>
<th>Objective</th>
<th>SumInf</th>
<th>MaxInf</th>
<th>Seconds</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>4.5165428E-01</td>
<td>5.76E-02</td>
<td>5.8E-02</td>
<td>0.007</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>6.700705E-01</td>
<td>5.03E-05</td>
<td>5.0E-05</td>
<td>0.014</td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>2.765930E+00</td>
<td>6.25E-04</td>
<td>6.2E-04</td>
<td>0.020</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>2.710653E+00</td>
<td>1.55E-02</td>
<td>1.6E-02</td>
<td>0.026</td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td>4.016702E+00</td>
<td>1.44E-02</td>
<td>1.4E-02</td>
<td>0.032</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>4.865399E+00</td>
<td>2.88E-04</td>
<td>2.9E-04</td>
<td>0.038</td>
<td></td>
</tr>
<tr>
<td>3500</td>
<td>4.858826E+00</td>
<td>3.31E-03</td>
<td>3.3E-03</td>
<td>0.044</td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>1.106472E+01</td>
<td>1.53E-02</td>
<td>1.5E-02</td>
<td>0.050</td>
<td></td>
</tr>
<tr>
<td>4500</td>
<td>1.595505E+01</td>
<td>1.56E-06</td>
<td>1.6E-06</td>
<td>0.055</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>1.618715E+01</td>
<td>2.17E-05</td>
<td>2.2E-05</td>
<td>0.062</td>
<td></td>
</tr>
<tr>
<td>5500</td>
<td>1.618987E+01</td>
<td>3.45E-04</td>
<td>3.5E-04</td>
<td>0.067</td>
<td></td>
</tr>
<tr>
<td>6000</td>
<td>1.985940E+01</td>
<td>4.03E-04</td>
<td>4.0E-04</td>
<td>0.074</td>
<td></td>
</tr>
<tr>
<td>6500</td>
<td>1.624319E+01</td>
<td>5.64E-03</td>
<td>5.6E-03</td>
<td>0.079</td>
<td></td>
</tr>
<tr>
<td>7000</td>
<td>1.727653E+01</td>
<td>8.98E-05</td>
<td>9.0E-05</td>
<td>0.086</td>
<td></td>
</tr>
<tr>
<td>7500</td>
<td>1.727033E+01</td>
<td>3.03E-03</td>
<td>3.0E-03</td>
<td>0.091</td>
<td></td>
</tr>
<tr>
<td>7840</td>
<td>2.933167E-02</td>
<td>0.00E+00</td>
<td>0.0E+00</td>
<td>0.097</td>
<td></td>
</tr>
</tbody>
</table>

LGO then reports the termination status, in this case globally optimal, together with the solver resource time. The resource time is also disaggregated by the total time spent performing function evaluations and the number of milliseconds (ms) spent for each function evaluation.

--- LGO Exit: Terminated by solver - Global solution
       0.047 LGO Secs (0.015 Eval Secs, 0.001 ms/eval)

A local solver such as CONOPT can be called to compute marginal values. To invoke a postsolve using CONOPT, the user specifies the callConopt option with a positive value, indicating the number of seconds CONOPT is given to solve. See the LGO option section for further details.

4 Illustrative References


1 Introduction

GAMS/LINDO finds guaranteed globally optimal solutions to general nonlinear problems with continuous and/or discrete variables. GAMS/LINDO supports most mathematical functions, including functions that are nonsmooth, such as abs(x) and or even discontinuous, such as floor(x). Nonlinear solvers employing methods like successive linear programming (SLP) or generalized reduced gradient (GRG) return a local optimal solution to an NLP problem. However, many practical nonlinear models are non-convex and have more than one local optimal solution. In some applications, the user may want to find a global optimal solution.

The LINDO global optimization procedure(GOP) employs branch-and-cut methods to break an NLP model down into a list of subproblems. Each subproblem is analyzed and either a) is shown to not have a feasible or optimal solution, or b) an optimal solution to the subproblem is found, e.g., because the subproblem is shown to be convex, or c) the subproblem is further split into two or more subproblems which are then placed on the list. Given appropriate tolerances, after a finite, though possibly large number of steps a solution provably global optimal to tolerances is returned. Traditional nonlinear solvers can get stuck at suboptimal, local solutions. This is no longer the case when using the global solver.

GAMS/LINDO can automatically linearize a number of nonlinear relationships, such as max(x,y), through the addition of constraints and integer variables, so the transformed linearized model is mathematically equivalent to the original nonlinear model. Keep in mind, however, that each of these strategies will require additional computation time. Thus, formulating
models, so they are convex and contain a single extremum, is desirable. In order to decrease required computing power and
time it is also possible to disable the global solver and use GAMS/LINDO like a regular nonlinear solver.

GAMS/LINDO has a multistart feature that restarts the standard (non-global) nonlinear solver from a number of intelligently
generated points. This allows the solver to find a number of locally optimal points and report the best one found. This
alternative can be used when global optimization is costly. A user adjustable parameter controls the maximum number of
multistarts to be performed.

LINDO automatically detects problem type and uses an appropriate solver, e.g., if you submit an LP model to LINDO, it
will be solved as an LP at LP speed, regardless of what you said in the "solve using” statement. With the NLP parameter
NLP QUADCHK turned on, LINDO can detect hidden quadratic expressions and automatically recognize convex QCPs, as
well as second-order cones (SOCP), like in Value-at-Risk models, allowing dramatically faster solution times via the barrier
solver. When such models have integer variables, LINDO would use the barrier solver to solve all subproblems leading to
significantly improved solution times when compared to the case with the standard NLP solver.

1.1 Licensing and software requirements

In order to use GAMS/LINDOGlobal, users need a GAMS/LINDOGlobal license. Additionally a GAMS/CONOPT license
is required for solving nonlinear subproblems. The GAMS/LINDOGlobal license places upper limits on model size of 3,000
variables and 2,000 constraints.

To use GAMS/LINDO, no additional license is required. Also, there is no upper limit on the model size with GAMS/LINDO
and, in addition, it allows to solve stochastic models (see section Stochastic Programming (SP) in GAMS/Lindo ).

Neither the GAMS/LINDO not the GAMS/LINDOGlobal license does include the Barrier solver option. LINDO would be
able to use the barrier solver when the user has a separate license for the GAMS/MOSEK barrier solver.

1.2 Running GAMS/LINDO

GAMS/LINDO is capable of solving models of the following types: LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP
and MINLP. If GAMS/LINDO is not specified as the default solver for these models, it can be invoked by issuing one of the
following command before the solve statement:

```plaintext
option xxx=lindo;
option xxx=lindoglobal;
```

where xxx is one of: LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP, or MINLP.

You can also find global optima to math programs with equilibrium or complementarity constraints, type MPEC, by using
the GAMS/NLPEC translator in conjunction with LINDO. You use NLPEC to translate complementarities into standard
mathematical statements, e.g. h’y = 0, and then use LINDO as the DNLP(Discontinuous Nonlinear) solver to solve the
translated model. The following little GAMS model illustrates:

```plaintext
$TITLE simple mpec example
variable f, x1, x2, y1, y2; positive
variable y1; y2.lo = -1; y2.up = 1;

equations cost, g, h1, h2;

cost.. f =E= x1 + x2;
g.. sqr(x1) + sqr(x2) =L= 1;
h1.. x1 =G= y1 - y2 + 1;
h2.. x2 + y2 =N= 0;

* declare h and y complementary
model example / cost, g, h1.y1, h2.y2 /;
```
option mpec=nlpec;
option dnlp=lindo;
solve example using mpec min f;

2 Supported nonlinear functions

GAMS/LINDO supports most nonlinear functions in global mode, including +, -, *, /, floor, modulo, sign, min, max, sqr, exp, power, ln, log, sqrt, abs, cos, sin, tan, cosh, sinh, tanh, arccos, arcsin, arctan and logic expressions AND, OR, NOT, and IF. Be aware that using highly nonconvex functions may lead to long solve times.

3 GAMS/LINDO output

The log output below is obtained for the NLP model mhw4d.gms from the GAMS model library using LINDO’s global solver.

LINDO 24Nov11 23.8.0 WIN 30200.30202 VS8 x86/MS Windows

LINDO Driver

Lindo API version 7.0.1.372 built on Nov 3 2011 21:49:01
Barrier Solver Version 6.0.0.114, Nonlinear Solver Version 3.15B
Platform Windows x86

Number of constraints: 3 le: 0, ge: 0, eq: 3, rn: 0 (ne:0)
Number of variables : 5 lb: 0, ub: 0, fr: 5, bx: 0 (fx:0)
Number of nonzeroes : 8 density=0.0053(%)  
Nonlinear variables : 5
Nonlinear constraints: 4
Nonlinear nonzeroes : 5+5

Starting global optimization ...

Number of nonlinear functions/operators: 3

Starting GOP presolve ...
First Call Local Solver
Find local solution, objvalue = 27.871905

Pre-check unboundedness
Computing reduced bound...
Searching for a better solution...

Starting reformulation ...

<table>
<thead>
<tr>
<th>Model</th>
<th>Input</th>
<th>Operation</th>
<th>Atomic</th>
<th>Convex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of variables : 5</td>
<td>6</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Number of constraints: 3</td>
<td>4</td>
<td>18</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>integer variables : 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
nonlinear variables : 5 5 9 0

Starting global search ...
Initial upper bound on objective: +2.931083e-002
Initial lower bound on objective: -3.167052e+022

<table>
<thead>
<tr>
<th>#NODEs</th>
<th>BOXES</th>
<th>LOWER BOUND</th>
<th>UPPER BOUND</th>
<th>RGAP</th>
<th>TIME(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-3.167052e+022</td>
<td>+2.931083e-002</td>
<td>1.0e+000</td>
<td>0 (*N)</td>
</tr>
<tr>
<td>19</td>
<td>17</td>
<td>-2.136461e+000</td>
<td>+2.931083e-002</td>
<td>1.0e+000</td>
<td>0 (*I)</td>
</tr>
<tr>
<td>22</td>
<td>20</td>
<td>-1.848574e-001</td>
<td>+2.931083e-002</td>
<td>2.1e-001</td>
<td>0 (*I)</td>
</tr>
<tr>
<td>23</td>
<td>21</td>
<td>+2.416053e-003</td>
<td>+2.931083e-002</td>
<td>2.7e-002</td>
<td>0 (*F)</td>
</tr>
</tbody>
</table>

Terminating global search ...

Global optimum found
Objective value : 0.0293108307216
Best Bound : 0.00241605257558
Factors (ok,stb) : 522 (100.00,99.81)
Simplex iterations : 2503
Barrier iterations : 0
Nonlinear iterations : 433
Box iterations : 23
Total number of boxes : 21
Max. Depth : 5
First solution time (sec.) : 0
Best solution time (sec.) : 0
Total time (sec.) : 0

After determining the different kinds of nonlinear operators LINDO tries to linearize these within the presolving. When a feasible starting point is found the optimization starts and the log provides information about the progress. At the end it is reported if an optimum could be found and then the results as well as the used resources are summarized.

The following flags can be seen in the progress log:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(*FP)</td>
<td>found a new MIP solution with feasibility pump</td>
</tr>
<tr>
<td>(*SBB)</td>
<td>found a new MIP solution in tree reorder</td>
</tr>
<tr>
<td>(*SE)</td>
<td>found a new MIP solution in simple enumeration</td>
</tr>
<tr>
<td>(*AB)</td>
<td>found a new MIP solution in advanced branching</td>
</tr>
<tr>
<td>(*AH)</td>
<td>found a new MIP solution with advanced heuristics</td>
</tr>
<tr>
<td>(*C)</td>
<td>found a new MIP solution after cuts added</td>
</tr>
<tr>
<td>(*T)</td>
<td>found a new MIP solution on the top</td>
</tr>
<tr>
<td>(*SRH)</td>
<td>found a new MIP solution in simple rounding heuristics</td>
</tr>
<tr>
<td>(*SB)</td>
<td>found a new MIP solution in strong branching</td>
</tr>
<tr>
<td>(*K)</td>
<td>found a new MIP solution in knapsack enumerator</td>
</tr>
<tr>
<td>(*)</td>
<td>found a new MIP solution normal branching</td>
</tr>
<tr>
<td>(*?-)</td>
<td>found a new MIP solution with advanced heuristics (level$&gt;10$)</td>
</tr>
<tr>
<td>(*N)</td>
<td>found a new incumbent GOP solution</td>
</tr>
<tr>
<td>(*I)</td>
<td>stored a box with the incumbent solution into the GOP solution list</td>
</tr>
<tr>
<td>(*F)</td>
<td>determined the final GOP status</td>
</tr>
</tbody>
</table>
4 Summary of GAMS/LINDO Options

GAMS/LINDO offers a diverse range of user-adjustable parameters to control the behavior of its solvers. While the default values of these parameters work best for most purposes, there may be cases the users prefer to work with different settings for a subset of the available parameters. This section gives a list of available GAMS/LINDO parameters, categorized by type, along with their brief descriptions. A more detailed description is given in the section that follows.

4.1 GAMS/LINDO Options File

In order to set GAMS/LINDO options, you need to set up an option file `lindo.opt` or `lindoglobal.opt` in your GAMS project directory. You must indicate in the model that you want to use the option file by inserting before the solve statement, the line:

   `<modelname>.optfile = 1;`

where

   `<modelname>`

is the name of the model referenced in the model statement. The option file is in plain text format containing a single GAMS/LINDO option per line. Each option identifier is followed by its target value with space or tab characters separating them. The lines starting with `*` character are treated as comments.

A sample option file `lindo.opt` looks like below

   * Use(1) or Disable(0) global optimization for NLP/MINLP models
     USEGOP        0

   * Enable Multistart NLP solver
     NLP_SOLVER    9

   * Allow a maximum of 3 multistart attempts
     NLP_MAXLOCALSEARCH 3

   * Set an overall time limit of 200 secs.
     SOLVER_TIMLMT 200

5 Summary of GAMS/Lindo Options

5.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECOMPOSITION_TYPE</td>
<td>decomposition to be performed on a linear or mixed integer model</td>
<td>1</td>
</tr>
<tr>
<td>FIND_BLOCK</td>
<td>graph partitioning method to find block structures</td>
<td>0</td>
</tr>
<tr>
<td>INSTRUCT_SUBOUT</td>
<td>flag to specify how to deal with fixed variables in the instruction list</td>
<td>-1</td>
</tr>
<tr>
<td>MULTITHREAD_MODE</td>
<td>threading mode</td>
<td>-1</td>
</tr>
<tr>
<td>NUM_THREADS</td>
<td>number of parallel threads to be used</td>
<td>GAMS Threads</td>
</tr>
<tr>
<td>PROFILER_LEVEL</td>
<td>specifies the profiler level to break down the total cpu time into.</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_FEASTOL</td>
<td>feasibility tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>SOLVER_IUSOL</td>
<td>flag for computing basic solution for infeasible model</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_OPTTOL</td>
<td>dual feasibility tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>SOLVER_PRE_ELIM_FILL</td>
<td>fill-in introduced by the eliminations during pre-solve</td>
<td>1000</td>
</tr>
<tr>
<td>SOLVER_RESTART</td>
<td>starting basis flag</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_TIMLMT</td>
<td>time limit in seconds for continuous solver</td>
<td>GAMS ResLim</td>
</tr>
</tbody>
</table>

### 5.2 LP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP_PRELEVEL</td>
<td>controls the amount and type of LP pre-solving</td>
<td>126</td>
</tr>
<tr>
<td>PROB_TO_SOLVE</td>
<td>controls whether the explicit primal or dual form of the given LP problem will be solved</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_CONCURRENT_OPTMODE</td>
<td>controls if simplex and interior-point optimizers will run concurrently</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_CUTOFFVAL</td>
<td>solver will exit if optimal solution is worse than this</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_IPMSOL</td>
<td>basis crossover flag for barrier solver</td>
<td>0</td>
</tr>
<tr>
<td>SOLVER_USECUTOFFVAL</td>
<td>flag for using cutoff value</td>
<td>0</td>
</tr>
<tr>
<td>SPLEX_DPRICING</td>
<td>pricing option for dual simplex method</td>
<td>-1</td>
</tr>
<tr>
<td>SPLEX_DUAL_PHASE</td>
<td>controls the dual simplex strategy</td>
<td>0</td>
</tr>
<tr>
<td>SPLEX_ITRLMT</td>
<td>simplex iteration limit</td>
<td>infinity</td>
</tr>
<tr>
<td>SPLEX_PPRICING</td>
<td>pricing option for primal simplex method</td>
<td>-1</td>
</tr>
<tr>
<td>SPLEX_REFACFRQ</td>
<td>number of simplex iterations between two consecutive basis re-factorizations</td>
<td>100</td>
</tr>
<tr>
<td>SPLEX_SCALE</td>
<td>scaling flag</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.3 IPM Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPM_BASIS_REL_TOL_S</td>
<td>maximum relative dual bound violation allowed in an optimal basic solution</td>
<td>1e-12</td>
</tr>
<tr>
<td>IPM_BASIS_TOL_S</td>
<td>maximum absolute dual bound violation in an optimal basic solution</td>
<td>1e-7</td>
</tr>
<tr>
<td>IPM_BASIS_TOL_X</td>
<td>maximum absolute primal bound violation allowed in an optimal basic solution</td>
<td>1e-7</td>
</tr>
<tr>
<td>IPM_BI_LU_TOL_REL_PIV</td>
<td>relative pivot tolerance used in the LU factorization in the basis identification procedure</td>
<td>1e-2</td>
</tr>
<tr>
<td>IPM_CHECK_CONVEXITY</td>
<td>flag to check convexity of a quadratic program using barrier solver</td>
<td>1</td>
</tr>
<tr>
<td>IPM_CO_TOL_INFEAS</td>
<td>maximum bound infeasibility tolerance for Conic solver</td>
<td>1e-10</td>
</tr>
<tr>
<td>IPM_MAX_ITERATIONS</td>
<td>ipm iteration limit</td>
<td>1000</td>
</tr>
<tr>
<td>IPM_NUM_THREADS</td>
<td>number of threads to run the interiorpoint optimizer on</td>
<td>1</td>
</tr>
<tr>
<td>IPM_OFF_COL_TRH</td>
<td>extent for detecting the offending columns in the Jacobian of the constraint matrix</td>
<td>40</td>
</tr>
<tr>
<td>IPM_TOL_DFEAS</td>
<td>dual feasibility tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>IPM_TOL_DSAFE</td>
<td>controls the initial dual starting point</td>
<td>1</td>
</tr>
<tr>
<td>IPM_TOL_INFEAS</td>
<td>infeasibility tolerance</td>
<td>1e-10</td>
</tr>
<tr>
<td>IPM_TOL_MU_RED</td>
<td>relative complementarity gap tolerance</td>
<td>1e-16</td>
</tr>
<tr>
<td>IPM_TOL_PATH</td>
<td>how close to follow the central path</td>
<td>1e-8</td>
</tr>
<tr>
<td>IPM_TOL_PFEAS</td>
<td>primal feasibility tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>IPM_TOL_PSAFE</td>
<td>controls the initial primal starting point</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.4 MIP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIP_ABSCUTTOL</td>
<td>MIP absolute cut tolerance</td>
<td>~1.0</td>
</tr>
<tr>
<td>MIP_ABSOPTTOL</td>
<td>MIP absolute optimality tolerance</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td>MIP_ADDCUTOBJTOL</td>
<td>required objective improvement to continue generating cuts</td>
<td>1.5625e-5</td>
</tr>
<tr>
<td>MIP_ADDCUTPER</td>
<td>percentage of constraint cuts that can be added</td>
<td>0.75</td>
</tr>
<tr>
<td>MIP_ADDCUTPER_TREE</td>
<td>percentage of constraint cuts that can be added at child nodes</td>
<td>0.75</td>
</tr>
<tr>
<td>MIP_AGGCUTLIM_TOP</td>
<td>max number of constraints involved in derivation of aggregation cut at root node</td>
<td>~1</td>
</tr>
<tr>
<td>MIP_AGGCUTLIM_TREE</td>
<td>max number of constraints involved in derivation of aggregation cut at tree nodes</td>
<td>~1</td>
</tr>
<tr>
<td>MIP_ANODES_SWITCH_DF</td>
<td>threshold on active nodes for switching to depth-first search</td>
<td>50000</td>
</tr>
<tr>
<td>MIP_AOPTTIMLIM</td>
<td>time in seconds beyond which the relative optimality tolerance will be applied</td>
<td>100</td>
</tr>
<tr>
<td>MIP_BIGM_FOR_INTTOL</td>
<td>threshold for which coefficient of a binary variable would be considered as big-M</td>
<td>1e8</td>
</tr>
<tr>
<td>MIP_BRANCHDIR</td>
<td>first branching direction</td>
<td>0</td>
</tr>
<tr>
<td>MIP_BRANCHRULE</td>
<td>rule for choosing the variable to branch</td>
<td>0</td>
</tr>
<tr>
<td>MIP_BRANCH_LIMIT</td>
<td>limit on the total number of branches to be created during branch and bound</td>
<td>~1</td>
</tr>
<tr>
<td>MIP_BRANCH_PRIO</td>
<td>controls how variable selection priorities are set and used</td>
<td>0</td>
</tr>
<tr>
<td>MIP_CUTDEPTH</td>
<td>threshold value for the depth of nodes in the branch and bound tree</td>
<td>8</td>
</tr>
<tr>
<td>MIP_CUTFREQ</td>
<td>frequency of invoking cut generation at child nodes</td>
<td>10</td>
</tr>
<tr>
<td>MIP_CUTLEVEL_TOP</td>
<td>combination of cut types to try at the root node when solving a MIP</td>
<td>57342</td>
</tr>
<tr>
<td>MIP_CUTLEVEL_TREE</td>
<td>combination of cut types to try at child nodes in the branch and bound tree when solving a MIP</td>
<td>53246</td>
</tr>
<tr>
<td>MIP_CUTOFFOBJ</td>
<td>defines limit for branch and bound</td>
<td>1e30</td>
</tr>
<tr>
<td>MIP_CUTTIMLIM</td>
<td>time to be spent in cut generation</td>
<td>~1</td>
</tr>
<tr>
<td>MIP_DELTA</td>
<td>near-zero value used in linearizing nonlinear expressions</td>
<td>1e-6</td>
</tr>
<tr>
<td>MIP_DUAL_SOLUTION</td>
<td>flag for computing dual solution of LP relaxation</td>
<td>0</td>
</tr>
<tr>
<td>MIP_FP_ITRLIM</td>
<td>iteration limit for feasibility pump heuristic</td>
<td>500</td>
</tr>
<tr>
<td>MIP_FP_MODE</td>
<td>mode for the feasibility pump heuristic</td>
<td>~1</td>
</tr>
<tr>
<td>MIP_FP_OPT_METHOD</td>
<td>optimization and reoptimization method for feasibility pump heuristic</td>
<td>0</td>
</tr>
<tr>
<td>MIP_FP_TIMLIM</td>
<td>time limit for feasibility pump heuristic</td>
<td>1800</td>
</tr>
<tr>
<td>MIP_FP_WEIGHT</td>
<td>weight of the objective function in the feasibility pump</td>
<td>1</td>
</tr>
<tr>
<td>MIP_GENERAL_MODE</td>
<td>general strategy in solving MIPs</td>
<td>0</td>
</tr>
<tr>
<td>MIP_HEUELEVEL</td>
<td>specifies heuristic used to find integer solution</td>
<td>3</td>
</tr>
<tr>
<td>MIP_HEUMINTIMLIM</td>
<td>minimum time in seconds to be spent in finding heuristic solutions</td>
<td>0</td>
</tr>
<tr>
<td>MIP_HEU_DROP_OBJ</td>
<td>flag for whether to use without OBJ heuristic</td>
<td>0</td>
</tr>
<tr>
<td>MIP_HEU_MODE</td>
<td>heuristic used in MIP solver</td>
<td>0</td>
</tr>
<tr>
<td>MIP_INTTOL</td>
<td>absolute integer feasibility tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>MIP_ITRLIM</td>
<td>iteration limit for branch and bound</td>
<td>infinity</td>
</tr>
<tr>
<td>MIP_KEEPINMEM</td>
<td>flag for keeping LP bases in memory</td>
<td>1</td>
</tr>
<tr>
<td>MIP_LBIGN</td>
<td>Big-M value used in linearizing nonlinear expressions</td>
<td>10000</td>
</tr>
<tr>
<td>MIP_LSOLTIMLIM</td>
<td>time limit until finding a new integer solution</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_MAKECUT_INACTIVE_COUNT</td>
<td>threshold for times a cut could remain active after successive reoptimization</td>
<td>10</td>
</tr>
<tr>
<td>MIP_MAXCUTPASS_TOP</td>
<td>number passes to generate cuts on the root node</td>
<td>200</td>
</tr>
<tr>
<td>MIP_MAXCUTPASS_TREE</td>
<td>number passes to generate cuts on the child nodes</td>
<td>2</td>
</tr>
<tr>
<td>MIP_MAXNONIMP_CUTPASS</td>
<td>number of passes allowed in cut-generation that does not improve current relaxation</td>
<td>3</td>
</tr>
<tr>
<td>MIP_MAXNUM_MIP_SOL_STORAGE</td>
<td>maximum number of k-best solutions to store</td>
<td>1</td>
</tr>
<tr>
<td>MIP_MINABSOBJSTEP</td>
<td>value to update cutoff value each time a mixed integer solution is found</td>
<td>0</td>
</tr>
<tr>
<td>MIP_NODESELRULE</td>
<td>specifies the node selection rule</td>
<td>0</td>
</tr>
<tr>
<td>MIP_NUM_THREADS</td>
<td>number of parallel threads to use by the parallel MIP solver</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PARA_FP</td>
<td>flag for whether to use parallelization on the feasibility pump heuristic</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PARA_FP_MODE</td>
<td>flag for the mode of parallel feasibility pump</td>
<td>0</td>
</tr>
<tr>
<td>MIP_PARA_INIT_NODE</td>
<td>number of initial nodes for MIP parallelization</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_PARA_ITRLIM</td>
<td>flag for iteration mode in MIP parallelization</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PARA_RND_ITRLMT</td>
<td>iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations</td>
<td>2.0</td>
</tr>
<tr>
<td>MIP_PARA_SUB</td>
<td>flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing</td>
<td>1</td>
</tr>
<tr>
<td>MIP_PEROPTTOL</td>
<td>MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds</td>
<td>1e-5</td>
</tr>
<tr>
<td>MIP_PERSPECTIVE_REFORM</td>
<td>flag for whether to use Persective Reformulation</td>
<td>1</td>
</tr>
<tr>
<td>MIP_POLISH_ALPHA_TARGET</td>
<td>proportion solutions in the pool to initiate a polishing-task at the current node</td>
<td>0.6</td>
</tr>
<tr>
<td>MIP_POLISH_MAX_BRANCH_COUNT</td>
<td>maximum number of branches to polish</td>
<td>2000</td>
</tr>
<tr>
<td>MIP_POLISH_NUM_BRANCH_NEXT</td>
<td>number of branches to polish in the next round</td>
<td>4000</td>
</tr>
<tr>
<td>MIP_PREHEU_DFE_VSTLIM</td>
<td>limit for the variable visit in depth first enumeration</td>
<td>200</td>
</tr>
<tr>
<td>MIP_PREHEU_LEVEL</td>
<td>heuristic level for the prerelax solver</td>
<td>0</td>
</tr>
<tr>
<td>MIP_PREHEU_TC_ITRLIM</td>
<td>iteration limit for the two change heuristic</td>
<td>30000000</td>
</tr>
<tr>
<td>MIP_PREHEU_VAR_SEQ</td>
<td>sequence of the variable considered by the prerelax heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_PRELEVEL</td>
<td>controls the amount and type of MIP pre-solving at root node</td>
<td>3070</td>
</tr>
<tr>
<td>MIP_PRELEVEL_TREE</td>
<td>amount and type of MIP pre-solving at tree nodes</td>
<td>1214</td>
</tr>
<tr>
<td>MIP_PRE_ELIM_FILL</td>
<td>controls fill-in introduced by eliminations during pre-solve</td>
<td>100</td>
</tr>
<tr>
<td>MIP_PSEUDOCOST_RULE</td>
<td>specifies the rule in pseudocost computations for variable selection</td>
<td>0</td>
</tr>
<tr>
<td>MIP_PSEUDOCOST_WEIGHT</td>
<td>weight in pseudocost computations for variable selection</td>
<td>1.5625e-05</td>
</tr>
<tr>
<td>MIP_REDCOSTFIX_CUTOFF</td>
<td>cutoff value as a percentage of the reduced costs</td>
<td>0.9</td>
</tr>
</tbody>
</table>
### MIP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIP_REDCOSTFIX,CUTOFF_TREE</td>
<td>cutoff value as a percentage of the reduced costs at tree nodes</td>
<td>0.9</td>
</tr>
<tr>
<td>MIP_RELINTTOL</td>
<td>relative integer feasibility tolerance</td>
<td>8e-6</td>
</tr>
<tr>
<td>MIP_RELOPTTOL</td>
<td>MIP relative optimality tolerance</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>MIP_REOPT</td>
<td>optimization method to use when doing reoptimization</td>
<td>0</td>
</tr>
<tr>
<td>MIP_SCALING_BOUND</td>
<td>maximum difference between bounds of an integer variable for enabling scaling</td>
<td>10000</td>
</tr>
<tr>
<td>MIP_SOLVERTYPE</td>
<td>optimization method to use when solving mixed-integer models</td>
<td>0</td>
</tr>
<tr>
<td>MIP_STRONGBRANCHDONUM</td>
<td>minimum number of variables to try the strong branching on</td>
<td>3</td>
</tr>
<tr>
<td>MIP_STRONGBRANCHLEVEL</td>
<td>depth from the root in which strong branching is used</td>
<td>10</td>
</tr>
<tr>
<td>MIP_SWITCHFAC_SIM_IPM_TIME</td>
<td>factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_TIMLIM</td>
<td>time limit in seconds for integer solver</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>MIP_TOPOPT</td>
<td>optimization method to use when there is no previous basis</td>
<td>0</td>
</tr>
<tr>
<td>MIP_TREEREORDERLEVEL</td>
<td>tree reordering level</td>
<td>10</td>
</tr>
<tr>
<td>MIP_TREEREORDERMODE</td>
<td>tree reordering mode</td>
<td>1</td>
</tr>
<tr>
<td>MIP_USECUTOFFOBJ</td>
<td>flag for using branch and bound limit</td>
<td>1</td>
</tr>
<tr>
<td>MIP_USE_CUTS_HEU</td>
<td>controls if cut generation is enabled during MIP heuristics</td>
<td>-1</td>
</tr>
<tr>
<td>MIP_USE_ENUM_HEU</td>
<td>frequency of enumeration heuristic</td>
<td>4</td>
</tr>
<tr>
<td>MIP_USE_INT_ZERO_TOL</td>
<td>controls if all MIP calculations would be based on absolute integer feasibility tolerance</td>
<td>0</td>
</tr>
</tbody>
</table>

### NLP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP_AUTODERIV</td>
<td>defining type of computing derivatives</td>
<td>0</td>
</tr>
<tr>
<td>NLP_AUTOHESS</td>
<td>flag for using Second Order Automatic Differentiation for solving NLP</td>
<td>0</td>
</tr>
<tr>
<td>NLP_DERIV,DIFFTYPE</td>
<td>flag indicating the technique used in computing derivatives with finite differences</td>
<td>0</td>
</tr>
<tr>
<td>NLP_FEASCHK</td>
<td>how to report results when solution satisfies tolerance of scaled but not original model</td>
<td>1</td>
</tr>
<tr>
<td>NLP_FEASTOL</td>
<td>feasibility tolerance for nonlinear constraints</td>
<td>1e-6</td>
</tr>
<tr>
<td>NLP_INF</td>
<td>numeric infinity for nonlinear models</td>
<td>1e30</td>
</tr>
<tr>
<td>NLP_IPM2GRG</td>
<td>switch from IPM solver to GRG solver when IPM fails due to numerical errors</td>
<td>1</td>
</tr>
<tr>
<td>NLP_ITERS_PER_LOGLINE</td>
<td>number of nonlinear iterations to elapse before next progress message</td>
<td>10</td>
</tr>
<tr>
<td>NLP_ITRLMT</td>
<td>nonlinear iteration limit</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>NLP_LINEARZ</td>
<td>extent to which the solver will attempt to linearize nonlinear models</td>
<td>0</td>
</tr>
<tr>
<td>NLP_MAXLOCALSEARCH</td>
<td>maximum number of local searches</td>
<td>5</td>
</tr>
<tr>
<td>NLP_MAXLOCALSEARCH_TREE</td>
<td>maximum number of multistarts</td>
<td>1</td>
</tr>
<tr>
<td>NLP_MAX_RETRY</td>
<td>maximum number refinement retries to purify the final NLP solution</td>
<td>5</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>NLP_MSW_EUDIST_THRES</td>
<td>euclidean distance threshold in multistart search</td>
<td>0.001</td>
</tr>
<tr>
<td>NLP_MSW_FILTMODE</td>
<td>filtering mode to exclude certain domains during sampling in multistart search</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_MAXNOIMP</td>
<td>maximum number of consecutive populations to generate without any improvements</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_MAXPOP</td>
<td>maximum number of populations to generate in multistart search</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_MAXREF</td>
<td>maximum number of reference points to generate trial points in multistart search</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_NORM</td>
<td>norm to measure the distance between two points in multistart search</td>
<td>2</td>
</tr>
<tr>
<td>NLP_MSW_NUM_THREADS</td>
<td>number of parallel threads to be used when solving an NLP model with the multistart solver</td>
<td>1</td>
</tr>
<tr>
<td>NLP_MSW_OVERLAP_RATIO</td>
<td>rate of replacement in successive populations</td>
<td>0.1</td>
</tr>
<tr>
<td>NLP_MSW_POXDIST_THRES</td>
<td>penalty function neighborhood threshold in multistart search</td>
<td>0.01</td>
</tr>
<tr>
<td>NLP_MSW_PREPMODE</td>
<td>preprocessing strategies in multistart solver</td>
<td>-1</td>
</tr>
<tr>
<td>NLP_MSW_RG_SEED</td>
<td>random number generator seed for the multistart solver</td>
<td>1019</td>
</tr>
<tr>
<td>NLP_MSW_SOLIDX</td>
<td>index of the multistart solution to be loaded</td>
<td>0</td>
</tr>
<tr>
<td>NLP_MSW_KKTRAD_FACTOR</td>
<td>KKT solution neighborhood factor in multistart search</td>
<td>0.85</td>
</tr>
<tr>
<td>NLP_MSW_XNULRAD_FACTOR</td>
<td>initial solution neighborhood factor in multistart search</td>
<td>0.5</td>
</tr>
<tr>
<td>NLP_PRELEVEL</td>
<td>controls the amount and type of NLP pre-solving</td>
<td>126</td>
</tr>
<tr>
<td>NLP_PSTEP_FINITEDIFF</td>
<td>value of the step length in computing the derivatives using finite differences</td>
<td>5e-7</td>
</tr>
<tr>
<td>NLP_QUADCHK</td>
<td>flag for checking if NLP is quadratic</td>
<td>1</td>
</tr>
<tr>
<td>NLP_REDTGTOl</td>
<td>tolerance for the gradients of nonlinear functions</td>
<td>1e-7</td>
</tr>
<tr>
<td>NLP_SOLVER</td>
<td>type of nonlinear solver</td>
<td>7</td>
</tr>
<tr>
<td>NLP_SOLVE_AS_LP</td>
<td>flag indicating if the nonlinear model will be solved as an LP</td>
<td>0</td>
</tr>
<tr>
<td>NLPSTALL_ITRLMT</td>
<td>iteration limit before a sequence of non-improving NLP iterations is declared as stalling</td>
<td>100</td>
</tr>
<tr>
<td>NLP_STARTPOINT</td>
<td>flag for using initial starting solution for NLP</td>
<td>1</td>
</tr>
<tr>
<td>NLP_SUBSOLVER</td>
<td>type of nonlinear subsolver</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_CRASH</td>
<td>flag for using simple crash routines for initial solution</td>
<td>0</td>
</tr>
<tr>
<td>NLP_USE_LINDO_CRASH</td>
<td>flag for using advanced crash routines for initial solution</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_SELCONES</td>
<td>flag for using selective constraint evaluations for solving NLP</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_SLP</td>
<td>flag for using sequential linear programming step directions for updating solution</td>
<td>1</td>
</tr>
<tr>
<td>NLP_USE_STEEPEDGE</td>
<td>flag for using steepest edge directions for updating solution</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.6 Global Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOP_ABSOPTTOL</td>
<td>absolute optimality tolerance</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td>GOP_ALGREFORMMD</td>
<td>algebraic reformulation rule for a GOP</td>
<td>18</td>
</tr>
<tr>
<td>GOP_BBSRCHMD</td>
<td>node selection rule in GOP branch-and-bound</td>
<td>1</td>
</tr>
<tr>
<td>GOP_BNDLIM</td>
<td>max magnitude of variable bounds used in GOP convexification</td>
<td>1e10</td>
</tr>
<tr>
<td>GOP_BOXTOL</td>
<td>minimal width of variable intervals</td>
<td>1e-6</td>
</tr>
<tr>
<td>GOP_BRANCHCHMD</td>
<td>direction to branch first when branching on a variable</td>
<td>5</td>
</tr>
</tbody>
</table>
### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOP_BRANCH_LIMIT</td>
<td>limit on the total number of branches to be created in GOP tree</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_CORELEVEL</td>
<td>strategy of GOP branch-and-bound</td>
<td>14</td>
</tr>
<tr>
<td>GOP_DECOMPPRTMD</td>
<td>decomposition point selection rule in GOP branch-and-bound</td>
<td>1</td>
</tr>
<tr>
<td>GOP_DELTATOL</td>
<td>delta tolerance in GOP convexification</td>
<td>1e-7</td>
</tr>
<tr>
<td>GOP_FLTTOL</td>
<td>floating-point tolerance</td>
<td>1e-10</td>
</tr>
<tr>
<td>GOP_HEU_MODE</td>
<td>heuristic used in global solver</td>
<td>0</td>
</tr>
<tr>
<td>GOP_ITRLIM</td>
<td>GOP iteration limit</td>
<td>infinity</td>
</tr>
<tr>
<td>GOP_ITRLIM_IPM</td>
<td>total barrier iteration limit summed over all branches in GOP</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_ITRLIM_NLP</td>
<td>total nonlinear iteration limit summed over all branches in GOP</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_ITRLIM_SIM</td>
<td>total simplex iteration limit summed over all branches in GOP</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_LIM_MODE</td>
<td>flag indicating which heuristic limit on sub-solver in GOP is based</td>
<td>1</td>
</tr>
<tr>
<td>GOP_LINEARZ</td>
<td>flag indicating if GOP exploits linearizable model</td>
<td>1</td>
</tr>
<tr>
<td>GOP_LSOBRANLIM</td>
<td>branch limit until finding a new nonlinear solution</td>
<td>-1</td>
</tr>
<tr>
<td>GOP_MAXWIDMD</td>
<td>maximum width flag for the global solution</td>
<td>0</td>
</tr>
<tr>
<td>GOP_MULTILINEAR</td>
<td>flag indicating if GOP exploits multi linear feature</td>
<td>1</td>
</tr>
<tr>
<td>GOP_NUM_THREADS</td>
<td>number of parallel threads to be used when solving a nonlinear model with the global optimizat</td>
<td>1</td>
</tr>
<tr>
<td>GOP_OPTCHKMD</td>
<td>criterion used to certify the global optimality</td>
<td>2</td>
</tr>
<tr>
<td>GOP_OPT_MODE</td>
<td>mode for GOP optimization</td>
<td>1</td>
</tr>
<tr>
<td>GOP_POSTLEVEL</td>
<td>amount and type of GOP postsolving</td>
<td>6</td>
</tr>
<tr>
<td>GOP_PRELEVEL</td>
<td>amount and type of GOP presolving</td>
<td>30</td>
</tr>
<tr>
<td>GOP_RELBRNDMD</td>
<td>reliable rounding in the GOP branch-and-bound</td>
<td>0</td>
</tr>
<tr>
<td>GOP_RELOPTTOL</td>
<td>relative optimality tolerance</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>GOP_SUBOUT_MODE</td>
<td>substituting out fixed variables</td>
<td>1</td>
</tr>
<tr>
<td>GOP_TIMLIM</td>
<td>time limit in seconds for GOP branch-and-bound</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>GOP_USEBNDLIM</td>
<td>max magnitude of variable bounds flag for GOP convexification</td>
<td>2</td>
</tr>
<tr>
<td>GOP_WIDTOL</td>
<td>maximal width of variable intervals</td>
<td>1e-4</td>
</tr>
<tr>
<td>USEGOP</td>
<td>use global optimization</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.7 SP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE_ORDER_BY_STAGE</td>
<td>order nontemporal models or not</td>
<td>1</td>
</tr>
<tr>
<td>REPORTEVSSL</td>
<td>solve and report the expected value solution</td>
<td>0</td>
</tr>
<tr>
<td>SAMP_CDSINC</td>
<td>correlation matrix diagonal shift increment</td>
<td>1e-6</td>
</tr>
<tr>
<td>SAMP_NCM_CUTOBJ</td>
<td>objective cutoff (target) value to stop the nearest correlation matrix (NCM)</td>
<td>1e-30</td>
</tr>
<tr>
<td>SAMP_NCM_DSTORAGE</td>
<td>flag to enable or disable sparse mode in NCM computations</td>
<td>-1</td>
</tr>
<tr>
<td>SAMP_NCM_ITERLIM</td>
<td>iteration limit for NCM method</td>
<td>100</td>
</tr>
<tr>
<td>SAMP_NCM_METHOD</td>
<td>bitmask to enable methods for solving the nearest correlation matrix (NCM)</td>
<td>5</td>
</tr>
<tr>
<td>SAMP_NCM_OPTTOL</td>
<td>optimality tolerance for NCM method</td>
<td>1e-7</td>
</tr>
<tr>
<td>SAMP_SCALE</td>
<td>flag to enable scaling of raw sample data</td>
<td>0</td>
</tr>
<tr>
<td>STOC_ABSOPTTOL</td>
<td>absolute optimality tolerance (w.r.t lower and upper bounds on the true</td>
<td>GAMS OptCA</td>
</tr>
<tr>
<td></td>
<td>objective) to stop the solver</td>
<td></td>
</tr>
<tr>
<td>STOC_ADD_MPI</td>
<td>flag to use add-instructions mode when building deteq</td>
<td>0</td>
</tr>
<tr>
<td>STOC_ALD_dual_feastol</td>
<td>dual feasibility tolerance for ALD</td>
<td>1e-4</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>STOC_ALD_dual_steplen</td>
<td>dual step length for ALD</td>
<td>0.9</td>
</tr>
<tr>
<td>STOC_ALD_inner_iter_lim</td>
<td>inner loop iteration limit for ALD</td>
<td>1000</td>
</tr>
<tr>
<td>STOC_ALD_outer_iter_lim</td>
<td>outer loop iteration limit for ALD</td>
<td>200</td>
</tr>
<tr>
<td>STOC_ALD_primal_feastol</td>
<td>primal feasibility tolerance for ALD</td>
<td>1e-4</td>
</tr>
<tr>
<td>STOC_ALD_primal_steplen</td>
<td>primal step length for ALD</td>
<td>0.5</td>
</tr>
<tr>
<td>STOC_autoaggr</td>
<td>flag to enable or disable autoaggregation</td>
<td>1</td>
</tr>
<tr>
<td>STOC_benchmark_scen</td>
<td>benchmark scenario to compare EVPI and EVMU against</td>
<td>-2</td>
</tr>
<tr>
<td>STOC_BIGM</td>
<td>big-M value for linearization and penalty functions</td>
<td>1e7</td>
</tr>
<tr>
<td>STOC_bucket_size</td>
<td>bucket size in Benders decomposition</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_calc_EVPI</td>
<td>flag to enable or disable calculation of EVPI</td>
<td>1</td>
</tr>
<tr>
<td>STOC_correlation_type</td>
<td>correlation type associated with correlation matrix</td>
<td>0</td>
</tr>
<tr>
<td>STOC_deqopt</td>
<td>method to solve the DETEQ problem</td>
<td>0</td>
</tr>
<tr>
<td>STOC_deteq_type</td>
<td>type of deterministic equivalent</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_DS_SUBFORM</td>
<td>subproblem formulation to use in DirectSearch</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_ELIM_FXVAR</td>
<td>flag to enable elimination of fixed variables from deteq MPI</td>
<td>1</td>
</tr>
<tr>
<td>STOC_INFBND</td>
<td>value to truncate infinite bounds at non-leaf nodes</td>
<td>1e9</td>
</tr>
<tr>
<td>STOC_ITER_LIM</td>
<td>iteration limit for stochastic solver</td>
<td>infinity</td>
</tr>
<tr>
<td>STOC_MAP_MPI2LP</td>
<td>flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements</td>
<td>0</td>
</tr>
<tr>
<td>STOC_MAX_NUMSCENS</td>
<td>maximum number of scenarios before forcing automatic sampling</td>
<td>40000</td>
</tr>
<tr>
<td>STOC_METHOD</td>
<td>stochastic optimization method to solve the model</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_NAMEDATA_LEVEL</td>
<td>name data level</td>
<td>1</td>
</tr>
<tr>
<td>STOC_NODELP_PRELEVEL</td>
<td>presolve level solving node-models</td>
<td>0</td>
</tr>
<tr>
<td>STOC_NSAMPLE_PER_STAGE</td>
<td>list of sample sizes per stage (starting at stage 2)</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_NSAMPLE_SPAR</td>
<td>common sample size per stochastic parameter</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_NSAMPLE_STAGE</td>
<td>common sample size per stage</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_NUM_THREADS</td>
<td>number of parallel threads</td>
<td>1</td>
</tr>
<tr>
<td>STOC_RELOPTTOL</td>
<td>relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>STOC_REL_DSTEPTOL</td>
<td>dual-step tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>STOC_REL_PSTEPTOL</td>
<td>primal-step tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>STOC_REOPT</td>
<td>reoptimization method to solve the node-models</td>
<td>0</td>
</tr>
<tr>
<td>STOC_RG_SEED</td>
<td>seed to initialize the random number generator</td>
<td>1031</td>
</tr>
<tr>
<td>STOC_SAMP_CONT_ONLY</td>
<td>flag to restrict sampling to continuous stochastic parameters only or not</td>
<td>1</td>
</tr>
<tr>
<td>STOC_SBD_MAXCUTS</td>
<td>max cuts to generate for master problem</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_SBD_NUMCANDID</td>
<td>maximum number of candidate solutions to generate at SBD root</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_SBD_OBJCUTFLAG</td>
<td>flag to enable objective cut in SBD master problem</td>
<td>1</td>
</tr>
<tr>
<td>STOC_SBD_OBJCUTVAL</td>
<td>RHS value of objective cut in SBD master problem</td>
<td>1e-30</td>
</tr>
<tr>
<td>STOC_SHARE_BEGSTAGE</td>
<td>stage beyond which node-models are shared</td>
<td>-1</td>
</tr>
<tr>
<td>STOC_TIME_LIM</td>
<td>time limit for stochastic solver</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>STOC_TOPOPT</td>
<td>optimization method to solve the root problem</td>
<td>0</td>
</tr>
<tr>
<td>STOC_VARCONTROL_METHOD</td>
<td>sampling method for variance reduction</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.8 Link Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKRANGE</td>
<td>calculate feasible range for variables</td>
<td>range.gdx</td>
</tr>
<tr>
<td>READPARAMS</td>
<td>read Lindo parameter file</td>
<td></td>
</tr>
<tr>
<td>WRITEDEMPI</td>
<td>write deterministic equivalent in MPI format</td>
<td></td>
</tr>
<tr>
<td>WRITEDEMPS</td>
<td>write deterministic equivalent in MPS format</td>
<td></td>
</tr>
<tr>
<td>WRITEMPI</td>
<td>write (S)MPI file of processed model</td>
<td></td>
</tr>
<tr>
<td>WRITEMPS</td>
<td>write (S)MPS file of processed model</td>
<td></td>
</tr>
</tbody>
</table>

### 6 Detailed Descriptions of GAMS/Lindo Options

**CHECKRANGE** *(string)*: calculate feasible range for variables

If this option is set, Lindo calculates the feasible range (determined by an upper and lower bound) for every variable in each equation while all other variables are fixed to their level. If set, the value of this option defines the name of the GDX file where the results are written to. For every combination of equation- and variable block there will be one symbol in the format `EquBlock_VarBlock(equ_Ind_1, ..., equ_Ind_M, var_Ind_1, ..., var_Ind_N, directions)`.

*(default = range.gdx)*

**CORE_ORDER_BY_STAGE** *(integer)*: order nontemporal models or not

Order nontemporal models or not.

*(default = 1)*

**DECOMPOSITION_TYPE** *(integer)*: decomposition to be performed on a linear or mixed integer model

This refers to the type of decomposition to be performed on a linear or mixed integer model.

*(default = 1)*

0 Solver decides which type of decomposition to use
1 Solver does not perform any decompositions and uses the original model
2 Attempt total decomposition
3 Decomposed model will have dual angular structure
4 Decomposed model will have block angular structure
5 Decomposed model will have both dual and block angular structure

**FIND_BLOCK** *(integer)*: graph partitioning method to find block structures

Specifies the graph partitioning method to find block structures.

*(default = 0)*

0 Use an edge-weight minimizing graph partitioning heuristic
1 Use a vertex-weight minimizing graph partitioning heuristic

**GOP_ABSOPTTOL** *(real)*: absolute optimality tolerance
Synonym: ABSOPTTOL
This value is the GOP absolute optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.
(default = GAMS OptCA)

**GOP_ALGREFORMMD (integer): algebraic reformulation rule for a GOP**

Synonym: ALGREFORMMD
This controls the algebraic reformulation rule for a GOP. The algebraic reformulation and analysis is very crucial in building a tight convex envelope to enclose the nonlinear/nonconvex functions. A lower degree of overestimation on convex envelopes helps increase the convergence rate to the global optimum.
(default = 18)
- +2 Rearrange and collect terms
- +4 Expand all parentheses
- +8 Retain nonlinear functions
- +16 Selectively expand parentheses

**GOP_BBSRCHMD (integer): node selection rule in GOP branch-and-bound**

Synonym: BBSRCHMD
This specifies the node selection rule for choosing between all active nodes in the GOP branch-and-bound tree when solving global optimization programs.
(default = 1)
- 0 Depth first search
- 1 Choose node with worst bound

**GOP_BNDLIM (real): max magnitude of variable bounds used in GOP convexification**

Synonym: BNDLIM
This value specifies the maximum magnitude of variable bounds used in the GOP convexification. Any lower bound smaller than the negative of this value will be treated as the negative of this value. Any upper bound greater than this value will be treated as this value. This helps the global solver focus on more productive domains.
(default = 1e10)

**GOP_BOXTOL (real): minimal width of variable intervals**

Synonym: BOXTOL
This value specifies the minimal width of variable intervals in a box allowed to branch.
(default = 1e−6)

**GOP_BRANCHMD (integer): direction to branch first when branching on a variable**

Synonym: BRANCHMD
This specifies the direction to branch first when branching on a variable. The branch variable is selected as the one that holds the largest magnitude in the measure.
(default = 5)
- 0 Absolute width
- 1 Locally relative width
- 2 Globally relative width
- 3 Globally relative distance from the convex minimum to the bounds
- 4 Absolute violation between the function and its convex envelope at the convex minimum
Relative violation between the function and its convex envelope at the convex minimum

**GOP_BRANCH_LIMIT (integer):** limit on the total number of branches to be created in GOP tree

*Synonym: BRANCH_LIMIT*

This is the limit on the total number of branches to be created during branch-and-bound in GOP tree. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible solution was found, it will be installed as the incumbent (best known) solution.

*(default = -1)*

**GOP_CORELEVEL (integer):** strategy of GOP branch-and-bound

*Synonym: CORELEVEL*

This controls the strategy of GOP branch-and-bound procedure.

*(default = 14)*

+ 2 LP convex relaxation
+ 4 NLP solving
+ 8 Box Branching

**GOP_DECOMPPTMD (integer):** decomposition point selection rule in GOP branch-and-bound

*Synonym: DECOMPPTMD*

This specifies the decomposition point selection rule. In the branch step of GOP branch-and-bound, a branch point M is selected to decompose the selected variable interval \([Lb, Ub]\) into two subintervals, \([Lb, M]\) and \([M, Ub]\).

*(default = 1)*

0 Mid-point
1 Local minimum or convex minimum

**GOP_DELTATOL (real):** delta tolerance in GOP convexification

*Synonym: DELTATOL*

This value is the delta tolerance in the GOP convexification. It is a measure of how closely the additional constraints added as part of convexification should be satisfied.

*(default = 1e−7)*

**GOP_FLTTOL (real):** floating-point tolerance

*Synonym: FLTTOL*

This value is the GOP floating-point tolerance. It specifies the maximum rounding errors in the floating-point computation.

*(default = 1e−10)*

**GOP_HEU_MODE (integer):** heuristic used in global solver

*Synonym: HEU_MODE*

This specifies the heuristic used in the global solver to find a good solution. Typically, if a heuristic is used, this will put more efforts in searching for good solutions, and less in bound tightening.

*(default = 0)*

0 No heuristic is used
1 A simple heuristic is used

**GOP_ITRLIM (real):** GOP iteration limit

*Synonym: ITRLIM*

This is the total iteration limit (including simplex, barrier and nonlinear iteration) summed over branches in GOP.
The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.
(default = infinity)

GOP_ITRLIM_IPM (real): total barrier iteration limit summed over all branches in GOP

Synonym: ITRLIM_IPM
This is the total barrier iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.
(default = -1)

GOP_ITRLIM_NLP (real): total nonlinear iteration limit summed over all branches in GOP

Synonym: ITRLIM_NLP
This is the total nonlinear iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.
(default = -1)

GOP_ITRLIM_SIM (real): total simplex iteration limit summed over all branches in GOP

Synonym: ITRLIM_SIM
This is the total simplex iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.
(default = -1)

GOP_LIM_MODE (integer): flag indicating which heuristic limit on sub-solver in GOP is based

Synonym: LIM_MODE
This is a flag indicating which heuristic limit on sub-solver in GOP is based.
(default = 1)

0 No limit
1 Time based limit
2 Iteration based limit
3 Both time and iteration based limit

GOP_LINEARZ (integer): flag indicating if GOP exploits linearizable model

This is a flag indicating if GOP exploits linearizable model.
(default = 1)

0 Exploit linearizable model
1 Do not exploit linearizable model

GOP_LSOLBRANLIM (integer): branch limit until finding a new nonlinear solution

Synonym: LSOLBRANLIM
This value controls the branch limit until finding a new nonlinear solution since the last nonlinear solution is found. The default value is -1, which means no branch limit is imposed.
(default = -1)

GOP_MAXWIDMD (integer): maximum width flag for the global solution

Synonym: MAXWIDMD
This is the maximum width flag for the global solution. The GOP branch-and-bound may continue contracting a box with an incumbent solution until its maximum width is smaller than GOP_WIDTOL.
(default = 0)

0 The maximum width criterion is suppressed
The maximum width criterion is performed

**GOP_MULTILINEAR (integer):** flag indicating if GOP exploits multi linear feature

This is a flag indicating if GOP exploits multi linear feature.

(default = 1)

0 Off

1 On

**GOP_NUM_THREADS (integer):** number of parallel threads to be used when solving a nonlinear model with the global optimization solver

This value specifies the number of parallel threads to be used when solving a nonlinear model with the global optimization solver.

(default = 1)

**GOP_OPTCHKMD (integer):** criterion used to certify the global optimality

Synonym: OPTCHKMD

This specifies the criterion used to certify the global optimality. When this value is 0, the absolute deviation of objective lower and upper bounds should be smaller than **GOP_ABSOPTTOL** at the global optimum. When its value is 1, the relative deviation of objective lower and upper bounds should be smaller than **GOP_RELOPTTOL** at the global optimum. 2 means either absolute or relative tolerance is satisfied at global optimum.

(default = 2)

**GOP_OPT_MODE (integer):** mode for GOP optimization

Synonym: OPT_MODE

This specifies the mode for GOP optimization.

(default = 1)

0 Global search for a feasible solution (thus a feasibility certificate)

1 Global search for an optimal solution

2 Global search for an unboundedness certificate

**GOP_POSTLEVEL (integer):** amount and type of GOP postsolving

Synonym: POSTLEVEL

This controls the amount and type of GOP post-solving. The default value is: 6 = 2+4 meaning to do both of the below options.

(default = 6)

+2 Apply LSgetBestBound

+4 Reoptimize variable bounds

**GOP_PRELEVEL (integer):** amount and type of GOP presolving

Synonym: PRELEVEL

This controls the amount and type of GOP pre-solving. The default value is: 30 = 2+4+8+16 meaning to do all of the below options.

(default = 30)

+2 Initial local optimization

+4 Initial linear constraint propagation

+8 Recursive linear constraint propagation

+16 Recursive nonlinear constraint propagation

**GOP_RELBRNMD (integer):** reliable rounding in the GOP branch-and-bound
Synonym: RELBRNDMD
This controls the reliable rounding rule in the GOP branch-and-bound. The global solver applies many suboptimizations to estimate the lower and upper bounds on the global optimum. A rounding error or numerical instability could unintentionally cut off a good solution. A variety of reliable approaches are available to improve the precision.
(default = 0)

+2 Use smaller optimality or feasibility tolerances and appropriate presolving options
+4 Apply interval arithmetic to reverify the solution feasibility

GOP_RELOPTTOL (real): relative optimality tolerance
Synonyms: OPTTOL RELOPTTOL
This value is the GOP relative optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.
(default = GAMS OptCR)

GOP_SUBOUT_MODE (integer): substituting out fixed variables
Synonym: SUBOUT_MODE
This is a flag indicating whether fixed variables are substituted out of the instruction list used in the global solver.
(default = 1)

0 Do not substitute out fixed variables
1 Substitute out fixed variables

GOP_TIMLIM (integer): time limit in seconds for GOP branch-and-bound
Synonym: TIMLIM
This is the time limit in seconds for GOP branch-and-bound.
(default = GAMS ResLim)

GOP_USEBNDLIM (integer): max magnitude of variable bounds flag for GOP convexification
Synonym: USEBNDLIM
This value is a flag for the parameter GOP_BNDLIM.
(default = 2)

0 Do not use the bound limit on the variables
1 Use the bound limit right at the beginning of global optimization
2 Use the bound limit after the initial local optimization if selected

GOP_WIDTOL (real): maximal width of variable intervals
Synonym: WIDTOL
This value specifies the maximal width of variable intervals for a box to be considered as an incumbent box containing an incumbent solution. It is used when GOP_MAXWIDMD is set at 1.
(default = 1e-4)

INSTRUCT_SUBOUT (integer): flag to specify how to deal with fixed variables in the instruction list
This is a flag indicating whether 1) fixed variables are substituted out of the instruction list, 2) performing numerical calculation on constant numbers and replacing with the results.
(default = -1)

-1 Solver decides
0 Substitutions will not be performed
1 Substitutions will be performed

**IPM\_BASIS\_REL\_TOL\_S** *(real)*: maximum relative dual bound violation allowed in an optimal basic solution

Maximum relative dual bound violation allowed in an optimal basic solution.

(default = 1e-12)

**IPM\_BASIS\_TOL\_S** *(real)*: maximum absolute dual bound violation in an optimal basic solution

Maximum absolute dual bound violation in an optimal basic solution.

(default = 1e-7)

**IPM\_BASIS\_TOL\_X** *(real)*: maximum absolute primal bound violation allowed in an optimal basic solution

Maximum absolute primal bound violation allowed in an optimal basic solution.

(default = 1e-7)

**IPM\_BI\_LU\_TOL\_REL\_PIV** *(real)*: relative pivot tolerance used in the LU factorization in the basis identification procedure

Relative pivot tolerance used in the LU factorization in the basis identification procedure.

(default = 1e-2)

**IPM\_CHECK\_CONVEXITY** *(integer)*: flag to check convexity of a quadratic program using barrier solver

This is a flag to check convexity of a quadratic program using barrier solver.

(default = 1)

- 1 Check convexity only without solving the model
- 0 Use barrier solver to check convexity
- 1 Do not use barrier solver to check convexity

**IPM\_CO\_TOL\_INFEAS** *(real)*: maximum bound infeasibility tolerance for Conic solver

Maximum bound infeasibility tolerance for Conic solver.

(default = 1e-10)

**IPM\_MAX\_ITERATIONS** *(integer)*: ipm iteration limit

Controls the maximum number of iterations allowed in the interior-point optimizer.

(default = 1000)

**IPM\_NUM\_THREADS** *(integer)*: number of threads to run the interiorpoint optimizer on

Number of threads to run the interiorpoint optimizer on. This value should be less than or equal to the actual number of processors or cores on a multi-core system.

(default = 1)

**IPM\_OFF\_COL\_TRH** *(integer)*: extent for detecting the offending columns in the Jacobian of the constraint matrix

Controls the extent for detecting the offending columns in the Jacobian of the constraint matrix. 0 means no offending columns will be detected. 1 means offending columns will be detected. In general, increasing the parameter value beyond the default value of 40 does not improve the result.

(default = 40)

**IPM\_TOL\_DFEAS** *(real)*: dual feasibility tolerance

Dual feasibility tolerance used for linear and quadratic optimization problems.

(default = 1e-8)

**IPM\_TOL\_DSAFE** *(real)*: controls the initial dual starting point

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the dual variables associated with constraint or variable bounds are very large, then it might be worthwhile to increase this value.
IPM.TOL.INFEAS (real): infeasibility tolerance
This is the tolerance to declare the model primal or dual infeasible using the interior-point optimizer. A smaller number means the optimizer gets more conservative about declaring the model infeasible.
(default = 1e-10)

IPM.TOL.MU.RED (real): relative complementarity gap tolerance
Relative complementarity gap tolerance.
(default = 1e-16)

IPM.TOL.PATH (real): how close to follow the central path
Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. For numerically unstable problems it might help to increase this parameter.
(default = 1e-8)

IPM.TOL.PFEAS (real): primal feasibility tolerance
Primal feasibility tolerance used for linear and quadratic optimization problems.
(default = 1e-8)

IPM.TOL.PSAFE (real): controls the initial primal starting point
Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.
(default = 1)

IPM.TOL.REL.STEP (real): relative step size to the boundary
Relative step size to the boundary for linear and quadratic optimization problems.
(default = 0.9999)

LP.PRELEVEL (integer): controls the amount and type of LP pre-solving
This controls the amount and type of LP pre-solving to be used.
(default = 126)
+2 Simple pre-solving
+4 Probing
+8 Coefficient reduction
+16 Elimination
+32 Dual reductions
+64 Use dual information
+512 Maximum pass

MIP.ABSCUTTOL (real): MIP absolute cut tolerance
This is the MIP absolute cut tolerance. If the value is less than or equal to zero, it will use the internal decided tolerance, otherwise it will use this value as the absolute tolerance for adding cuts.
(default = -1.0)

MIP.ABSOPTTOL (real): MIP absolute optimality tolerance
This is the MIP absolute optimality tolerance. Solutions must beat the incumbent by at least this absolute amount to become the new, best solution.
(default = GAMS OptCA)
MIP_ADDCUTOBJTOL (real): required objective improvement to continue generating cuts

This specifies the minimum required improvement in the objective function for the cut generation phase to continue generating cuts.
(default = 1.5625e-5)

MIP_ADDCUTPER (real): percentage of constraint cuts that can be added

This determines how many constraint cuts can be added as a percentage of the number of original rows in an integer programming model.
(default = 0.75)

MIP_ADDCUTPER_TREE (real): percentage of constraint cuts that can be added at child nodes

This determines how many constraint cuts can be added at child nodes as a percentage of the number of original rows in an integer programming model.
(default = 0.5)

MIP_AGGCUTLIM_TOP (integer): max number of constraints involved in derivation of aggregation cut at root node

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the root node. The default is .1, which means that the solver will decide.
(default = -1)

MIP_AGGCUTLIM_TREE (integer): max number of constraints involved in derivation of aggregation cut at tree nodes

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the tree nodes. The default is .1, which means that the solver will decide.
(default = -1)

MIP_ANODES_SWITCH_DF (integer): threshold on active nodes for switching to depth-first search

This specifies the threshold on active nodes for switching to depth-first search rule.
(default = 50000)

MIP_AOPTTIMLIM (integer): time in seconds beyond which the relative optimality tolerance will be applied

This is the time in seconds beyond which the relative optimality tolerance, MIP_PEROPTTOL will be applied.
(default = 100)

MIP_BIGM_FOR_INTTOL (real): threshold for which coefficient of a binary variable would be considered as big-M

This value specifies the threshold for which the coefficient of a binary variable would be considered as big-M (when applicable).
(default = 1e8)

MIP_BRANCHDIR (integer): first branching direction

This specifies the direction to branch first when branching on a variable.
(default = 0)

0 Solver decides
1 Always branch up first
2 Always branch down first

MIP_BRANCHRULE (integer): rule for choosing the variable to branch

This specifies the rule for choosing the variable to branch on at the selected node.
(default = 0)

0 Solver decides
1 Basis rounding with pseudo reduced costs
2 Maximum infeasibility
3 Pseudo reduced costs only
4 Maximum coefficient only
5 Previous branching only

**MIP_BRANCH_LIMIT (integer):** limit on the total number of branches to be created during branch and bound

This is the limit on the total number of branches to be created during branch-and-bound. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

(default = -1)

**MIP_BRANCH_PRIO (integer):** controls how variable selection priorities are set and used

This controls how variable selection priorities are set and used.

(default = 0)

0 If the user has specified priorities then use them Otherwise let LINDO API decide
1 If user has specified priorities then use them Overwrite users choices if necessary
2 If user has specified priorities then use them Otherwise do not use any priorities
3 Let LINDO API set the priorities and ignore any user specified priorities
4 Binaries always have higher priority over general integers

**MIP_CUTDEPTH (integer):** threshold value for the depth of nodes in the branch and bound tree

This controls a threshold value for the depth of nodes in the B&B tree, so cut generation will be less likely at those nodes deeper than this threshold.

(default = 8)

**MIP_CUTFREQ (integer):** frequency of invoking cut generation at child nodes

This controls the frequency of invoking cut generation at child nodes. The default value is 10, indicating that the MIP solver will try to generate cuts at every 10 nodes.

(default = 10)

**MIP_CUTLEVEL.TOP (integer):** combination of cut types to try at the root node when solving a MIP

This controls the combination of cut types to try at the root node when solving a MIP. Bit settings are used to enable the various cut types.

(default = 57342)
+2 GUB cover
+4 Flow cover
+8 Lifting
+16 Plant location
+32 Disaggregation
+64 Knapsack cover
+128 Lattice
+256 Gomory
+512 Coefficient reduction
+1024 GCD
+2048 Obj integrality
+4096 Basis Cuts
+8192 Cardinality Cuts
+16384 Disjunk Cuts
+32768 Soft Knapsack Cuts

**MIP.CUTLEVEL_TREE (integer):** combination of cut types to try at child nodes in the branch and bound tree when solving a MIP.

This controls the combination of cut types to try at child nodes in the B&B tree when solving a MIP.  
(default = 53246)

+2 GUB cover
+4 Flow cover
+8 Lifting
+16 Plant location
+32 Disaggregation
+64 Knapsack cover
+128 Lattice
+256 Gomory
+512 Coefficient reduction
+1024 GCD
+2048 Obj integrality
+4096 Basis Cuts
+8192 Cardinality Cuts
+16384 Disjunk Cuts
+32768 Soft Knapsack Cuts

**MIP.CUTOFFOBJ (real):** defines limit for branch and bound

If this is specified, then any part of the branch-and-bound tree that has a bound worse than this value will not be considered. This can be used to reduce the running time if a good bound is known.  
(default = 1e30)

**MIP.CUTTIMLIM (integer):** time to be spent in cut generation

This controls the total time to be spent in cut generation throughout the solution of a MIP. The default value is -1, indicating that no time limits will be imposed when generating cuts.  
(default = -1)

**MIP.DELTA (real):** near-zero value used in linearizing nonlinear expressions

This refers to a near-zero value used in linearizing nonlinear expressions.  
(default = 1e-6)

**MIP.DUAL_SOLUTION (integer):** flag for computing dual solution of LP relaxation

This flag controls whether the dual solution to the LP relaxation that yielded the optimal MIP solution will be computed or not.  
(default = 0)

0 Do not calculate dual solution for LP relaxation
1 Calculate dual solution for LP relaxation

**MIP.FP_ITRLIM (integer):** iteration limit for feasibility pump heuristic

This is the iteration limit in seconds for feasibility pump heuristic. A value of -1 means no iteration limit is
imposed.
(default = 500)

**MIP_FP_MODE (integer):** mode for the feasibility pump heuristic

Controls the mode for the feasibility pump heuristic.
(default = -1)

-1 Solver decides
0 Off
1 On until the first solution
2 Try to get more than one solution

**MIP_FP_OPT_METHOD (integer):** optimization and reoptimization method for feasibility pump heuristic

This specifies optimization and reoptimization method for feasibility pump heuristic.
(default = 0)

0 Solver decides
1 Primal simplex
2 Dual simplex
3 Barrier

**MIP_FP_TIMLIM (real):** time limit for feasibility pump heuristic

This is the time limit in seconds for feasibility pump heuristic. A value of -1 implies no time limit is imposed.
(default = 1800)

**MIP_FP_WEIGTH (real):** weight of the objective function in the feasibility pump

Controls the weight of the objective function in the feasibility pump.
(default = 1)

**MIP_GENERAL_MODE (integer):** general strategy in solving MIPs

This value specifies the general strategy in solving MIPs.
(default = 0)

0 Solver decides
+2 Disable all time-driven events for reproducibility of runs
+16 Disable cut generation before branching

**MIP_HEULEVEL (integer):** specifies heuristic used to find integer solution

This specifies the heuristic used to find the integer solution. Possible values are: 0: No heuristic is used. 1: A simple heuristic is used. Typically, this will find integer solutions only on problems with a certain structure. However, it tends to be fast. 2: This is an advanced heuristic that tries to find a "good" integer solution fast. In general, a value of 2 seems to not increase the total solution time and will find an integer solution fast on many problems. A higher value may find an integer solution faster, or an integer solution where none would have been found with a lower level. Try level 3 or 4 on "difficult" problems where 2 does not help. Higher values cause more time to be spent in the heuristic. The value may be set arbitrarily high. However, >20 is probably not worthwhile. **MIP_HEUMINTIMLIM** controls the time to be spent in searching heuristic solutions.
(default = 3)

**MIP_HEUMINTIMLIM (integer):** minimum time in seconds to be spent in finding heuristic solutions

This specifies the minimum time in seconds to be spent in finding heuristic solutions to the MIP model. **MIP_HEULEVEL** controls the heuristic used to find the integer solution.
(default = 0)
MIP\_HEU\_DROP\_OBJ (integer): flag for whether to use without OBJ heuristic

This is a flag for whether to use without OBJ heuristic.

(default = 0)

0 Off
1 On

MIP\_HEU\_MODE (integer): heuristic used in MIP solver

This controls the MIP heuristic mode.

(default = 0)

0 Solver decides when to stop the heuristic
1 Solver uses a pre-specified time limit to stop the heuristic.
2 Solver uses a pre-specified iteration limit to stop the heuristic

MIP\_INTTOL (real): absolute integer feasibility tolerance

An integer variable is considered integer feasible if the absolute difference from the nearest integer is smaller than this.

(default = 1e-6)

MIP\_ITRLIM (real): iteration limit for branch and bound

This is the total LP iteration limit summed over all branches for branch-and-bound. Range for The default value is -1, which means no iteration limit is imposed. If this iteration limit is reached, branch-and-bound will stop and the best feasible integer solution found will be installed as the incumbent (best known) solution.

(default = infinity)

MIP\_KEEPINMEM (integer): flag for keeping LP bases in memory

If this is set to 1, the integer pre-solver will try to keep LP bases in memory. This typically gives faster solution times, but uses more memory. Setting this parameter to 0 causes the pre-solver to erase bases from memory.

(default = 1)

0 Do not keep LP bases in memory
1 Keep LP bases in memory

MIP\_LBIGM (real): Big-M value used in linearizing nonlinear expressions

This refers to the Big-M value used in linearizing nonlinear expressions.

(default = 10000)

MIP\_LSOLTIMLIM (integer): time limit until finding a new integer solution

(default = -1)

MIP\_MAKECUT\_INACTIVE\_COUNT (integer): threshold for times a cut could remain active after successive reoptimization

This value specifies the threshold for the times a cut could remain active after successive reoptimization during branch-and-bound. If the count is larger than the specified level the solver will inactive the cut.

(default = 10)

MIP\_MAXCUTPASS\_TOP (integer): number passes to generate cuts on the root node

This controls the number passes to generate cuts on the root node. Each of these passes will be followed by a reoptimization and a new batch of cuts will be generated at the new solution.

(default = 200)

MIP\_MAXCUTPASS\_TREE (integer): number passes to generate cuts on the child nodes

This controls the number passes to generate cuts on the child nodes. Each of these passes will be followed by a
reoptimization and a new batch of cuts will be generated at the new solution.
(default = 2)

**MIP.MAXNONIMP.CUTPASS (integer):** number of passes allowed in cut-generation that does not improve current relaxation

This controls the maximum number of passes allowed in cut-generation that does not improve the current relaxation.
(default = 3)

**MIP.MAXNUM.MIP.SOL.STORAGE (integer):** maximum number of k-best solutions to store

This specifies the maximum number of k-best solutions to store. Possible values are positive integers.
(default = 1)

**MIP.MINABSOBJSTEP (real):** value to update cutoff value each time a mixed integer solution is found

This specifies the value to update the cutoff value each time a mixed integer solution is found.
(default = 0)

**MIP.NODESELRULE (integer):** specifies the node selection rule

This specifies the node selection rule for choosing between all active nodes in the branch-and-bound tree when solving integer programs. Possible selections are: 0: Solver decides (default). 1: Depth first search. 2: Choose node with worst bound. 3: Choose node with best bound. 4: Start with best bound. If no improvement in the gap between best bound and best integer solution is obtained for some time, switch to: if (number of active nodes < 10000) Best estimate node selection (5). else Worst bound node selection (2). 5: Choose the node with the best estimate, where the new objective estimate is obtained using pseudo costs. 6: Same as (4), but start with the best estimate.
(default = 0)

- 0 Solver decides
- 1 Depth first search
- 2 Choose node with worst bound
- 3 Choose node with best bound
- 4 Start with best bound
- 5 Choose the node with the best estimate
- 6 Same as 4 but start with the best estimate

**MIP.NUM.THREADS (integer):** number of parallel threads to use by the parallel MIP solver

This parameter specifies the number of parallel threads to use by the parallel MIP solver. Possible values are positive integers. The default is 1 implying that the parallel solver is disabled.
(default = 1)

**MIP.PARA.FP (integer):** flag for whether to use parallelization on the feasibility pump heuristic

This is a flag for whether to use parallelization on the feasibility pump heuristic.
(default = 1)

- 0 Off
- 1 On

**MIP.PARA.FP.MODE (integer):** flag for the mode of parallel feasibility pump

This is a flag for the mode of parallel feasibility pump.
(default = 0)

- 0 Terminate when all threads finish
1 Terminate as soon as the master thread finishes

**MIP_PARA_INIT_NODE** (*real*): number of initial nodes for MIP parallelization

This value specifies the number of initial nodes for MIP parallelization.

(default = -1)

**MIP_PARA_ITR_MODE** (*integer*): flag for iteration mode in MIP parallelization

This is a flag for iteration mode in MIP parallelization.

(default = 1)

- 0 Each thread terminates as soon as arrives iteration limit
- 1 Each thread terminates until all threads get iteration limit

**MIP_PARA_RND_ITRLMT** (*real*): iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations

This value specifies the iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations.

(default = 2.0)

**MIP_PARA_SUB** (*integer*): flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing

This is a flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing.

(default = 1)

- 0 Off
- 1 On

**MIP_PEROPTTOL** (*real*): MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds

This is the MIP relative optimality tolerance that will be in effect after T seconds following the start. The value T should be specified using the **MIP_AOPTTIMLIM** parameter.

(default = 1e-5)

**MIP_PERSPECTIVE_REFORM** (*integer*): flag for whether to use Perspective Reformulation

This is the flag for whether to use Perspective Reformulation.

(default = 1)

- 0 Off
- 1 On

**MIP_POLISH_ALPHA_TARGET** (*real*): proportion solutions in the pool to initiate a polishing-task at the current node

This value specifies the proportion solutions in the pool to initiate a polishing-task at the current node.

(default = 0.6)

**MIP_POLISH_MAX_BRANCH_COUNT** (*integer*): maximum number of branches to polish

This value specifies the maximum number of branches to polish.

(default = 2000)

**MIP_POLISH_NUM_BRANCH_NEXT** (*integer*): number of branches to polish in the next round

This value specifies the number of branches to polish in the next round.

(default = 4000)

**MIP_PREHEU_DFE_VSTLIM** (*integer*): limit for the variable visit in depth first enumeration

Limit for the variable visit in depth first enumeration.

(default = 200)

**MIP_PREHEU_LEVEL** (*integer*): heuristic level for the prerelax solver
The heuristic level for the prerelax solver.
(default = 0)

0  Nothing
1  One-change
2  One-change and two-change
3  Depth first enumeration

MIP\_PREHEU\_TC\_ITERLIM (integer): iteration limit for the two change heuristic

Iteration limit for the two change heuristic.
(default = 30000000)

MIP\_PREHEU\_VAR\_SEQ (integer): sequence of the variable considered by the prerelax heuristic

The sequence of the variable considered by the prerelax heuristic.
(default = -1)

-1  Backward
1  Forward

MIP\_PRELEVEL (integer): controls the amount and type of MIP pre-solving at root node

This controls the amount and type of MIP pre-solving at root node.
(default = 3070)

+2  Simple pre-solving
+4  Probing
+8  Coefficient reduction
+16  Elimination
+32  Dual reductions
+64  Use dual information
+128 Binary row presolving
+256 Row aggregation
+512 Coef Probe Lifting
+1024 Maximum pass
+2048 Similar row

MIP\_PRELEVEL\_TREE (integer): amount and type of MIP pre-solving at tree nodes

This controls the amount and type of MIP pre-solving at tree nodes.
(default = 1214)

+2  Simple pre-solving
+4  Probing
+8  Coefficient reduction
+16  Elimination
+32  Dual reductions
+64  Use dual information
+128 Binary row presolving
+256 Row aggregation
+512 Coef Probe Lifting
MIP_PRE_ELIM_FILL (integer): controls fill-in introduced by eliminations during pre-solve

This is a nonnegative value that controls the fill-in introduced by the eliminations during pre-solve. Smaller values could help when the total nonzeros in the presolved model is significantly more than the original model.
(default = 100)

MIP_PSEUDOCOST_RULE (integer): specifies the rule in pseudocost computations for variable selection

This specifies the rule in pseudocost computations for variable selection.
(default = 0)
0 Solver decides
1 Only use min pseudo cost
2 Only use max pseudo cost
3 Use quadratic score function and the pseudo cost weight
4 Same as 3 without quadratic score

MIP_PSEUDOCOST_WEIGHT (real): weight in pseudocost computations for variable selection

This specifies the weight in pseudocost computations for variable selection.
(default = 1.5625e-05)

MIP_REDCOSTFIX_CUTOFF (real): cutoff value as a percentage of the reduced costs

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic.
(default = 0.9)

MIP_REDCOSTFIX_CUTOFF_TREE (real): cutoff value as a percentage of the reduced costs at tree nodes

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic at tree nodes.
(default = 0.9)

MIP_RELINTTOL (real): relative integer feasibility tolerance

An integer variable is considered integer feasible if the difference between its value and the nearest integer value divided by the value of the nearest integer is less than this.
(default = 8e-6)

MIP_RELOPTTOL (real): MIP relative optimality tolerance

This is the MIP relative optimality tolerance. Solutions must beat the incumbent by at least this relative amount to become the new, best solution.
(default = GAMS OptCR)

MIP_REOPT (integer): optimization method to use when doing reoptimization

This specifies which optimization method to use when doing reoptimization from a given basis.
(default = 0)
0 Solver decides
1 Use primal method
2 Use dual simplex
3 Use barrier solver

MIP_SCALING_BOUND (integer): maximum difference between bounds of an integer variable for enabling scaling

This controls the maximum difference between the upper and lower bounds of an integer variable that will enable
the scaling in the simplex solver when solving a subproblem in the branch-and-bound tree.
(default = 10000)

**MIP_SOLVERTYPE (integer):** optimization method to use when solving mixed-integer models

This specifies the optimization method to use when solving mixed-integer models.

(default = 0)

0 Solver decides
1 Use Branch and Bound only
2 Use Enumeration and Knapsack solver only

**MIP_STRONGBRANCHDONUM (integer):** minimum number of variables to try the strong branching on

This value specifies the minimum number of variables, among all the candidates, to try the strong branching on.

(default = 3)

**MIP_STRONGBRANCHLEVEL (integer):** depth from the root in which strong branching is used

This specifies the depth from the root in which strong branching is used. The default value of 10 means that strong branching is used on a level of 1 to 10 measured from the root. Strong branching finds the real bound for branching on a given variable, which, in most cases, requires a solution of a linear program and may therefore also be quite expensive in computing time. However, if used on nodes close to the root node of the tree, it also gives a much better bound for that part of the tree and can therefore reduce the size of the branch-and-bound tree.

(default = 10)

**MIP_SWITCHFAC_SIM_IPM_TIME (real):** factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method

This specifies the (positive) factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method. A value of -1.0 means that no time limit is imposed.

(default = -1)

**MIP_TIMLIM (integer):** time limit in seconds for integer solver

This is the time limit in seconds for branch-and-bound. The default value is -1, which means no time limit is imposed. However, the value of SOLVER_TIMLIM will be applied to each continuous subproblem solve. If the value of this parameter is greater than 0, then the value of SOLVER_TIMLIM will be disregarded. If this time limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

(default = GAMS ResLim)

**MIP_TOPOPT (integer):** optimization method to use when there is no previous basis

This specifies which optimization method to use when there is no previous basis.

(default = 0)

0 Solver decides
1 Use primal method
2 Use dual simplex
3 Use barrier solver

**MIP_TREEREORDERLEVEL (integer):** tree reordering level

This specifies the tree reordering level.

(default = 10)

**MIP_TREEREORDERMODE (integer):** tree reordering mode

This specifies the tree reordering mode.

(default = 1)
1. Use tree reordering only for subproblems
2. Use tree reordering for subproblems and the main bnb loop only when LP status is infeasible
3. Do not use tree reordering
4. Use tree reordering based on MIP_TREEREORDERLEVEL

**MIP_USECUTOFFOBJ** *(integer)*: flag for using branch and bound limit

This is a flag for the parameter MIP_CUTOFFOBJ. If you do not want to lose the value of the parameter MIP_CUTOFFOBJ, this provides an alternative to disabling the cutoff objective.

(default = 1)

0. Do not use current cutoff value
1. Use current cutoff value

**MIP_USE_CUTS_HEU** *(integer)*: controls if cut generation is enabled during MIP heuristics

This flag controls if cut generation is enabled during MIP heuristics. The default is -1 (i.e. the solver decides).

(default = -1)

-1. Solver decides
0. Do not use cut heuristic
1. Use cut heuristic

**MIP_USE_ENUM_HEU** *(integer)*: frequency of enumeration heuristic

This specifies the frequency of enumeration heuristic.

(default = 4)

0. Off
1. Only at top (root) node without cuts
2. Both at top (root) and tree nodes without cuts
3. Same as 1 with cuts
4. Same as 2 with cuts

**MIP_USE_INT_ZERO_TOL** *(integer)*: controls if all MIP calculations would be based on absolute integer feasibility tolerance

This flag controls if all MIP calculations would be based on the integrality tolerance specified by MIP_INTTOL.

(default = 0)

0. Do not base MIP calculations on MIP_INTTOL
1. Base MIP calculations on MIP_INTTOL

**MULTITHREAD_MODE** *(integer)*: threading mode

This parameter controls the threading mode for solvers with multithreading support.

(default = -1)

-1. Solver decides
1. Try parallel mode but if it is not available try concurrent mode
2. Try parallel mode only
3. Try concurrent mode but if it is not available try parallel mode
4. Try concurrent mode only

**NLP_AUTODERIV** *(integer)*: defining type of computing derivatives

This is a flag to indicate if automatic differentiation is the method of choice for computing derivatives and select the type of differentiation.
(default = 0)
    0 Finite Differences approach will be used
    1 Forward type of Automatic Differentiation will be used
    2 Backward type of Automatic Differentiation will be used

**NLP.AUTOHESS (integer):** flag for using Second Order Automatic Differentiation for solving NLP

This is a flag to indicate if Second Order Automatic Differentiation will be performed in solving a nonlinear model. The second order derivatives provide an exact/precise Hessian matrix to the SQP algorithm, which may lead to less iterations and better solutions, but may also be quite expensive in computing time for some cases.

(default = 0)
    0 Do not use Second Order Automatic Differentiation
    1 Use Second Order Automatic Differentiation

**NLP.DERIV.DIFFTYPE (integer):** flag indicating the technique used in computing derivatives with finite differences

This is a flag indicating the technique used in computing derivatives with Finite Differences.

(default = 0)
    0 The solver decides
    1 Use forward differencing method
    2 Use backward differencing method
    3 Use center differencing method

**NLP.FEASCHK (integer):** how to report results when solution satisfies tolerance of scaled but not original model

This input parameter specifies how the NLP solver reports the results when an optimal or local-optimal solution satisfies the feasibility tolerance (**NLP.FEASTOL**) of the scaled model but not the original (descaled) one.

(default = 1)
    0 Perform no action accept the final solution
    1 Declare the model status as FEASIBLE if maximum violation in the unscaled model is not higher than 10 times NLP.FEASTOL
    2 Declare the model status as UNKNOWN if maximum violation in the unscaled model is higher than NLP.FEASTOL

**NLP.FEASTOL (real):** feasibility tolerance for nonlinear constraints

This is the feasibility tolerance for nonlinear constraints. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.

(default = 1e-6)

**NLP.INF (real):** numeric infinity for nonlinear models

Specifies the numeric infinity for nonlinear models. Possible values are positive real numbers. Smaller values could cause numerical problems.

nlp_ipm2grg This is a flag to switch from IPM solver to the standard NLP (GRG) solver when IPM fails due to numerical errors.

(default = 1e30)

**NLP.IPMP2GRG (integer):** switch from IPM solver to GRG solver when IPM fails due to numerical errors

(default = 1)
    0 Do not switch
    1 Switch
**NLP_ITERS_PER_LOGLINE** *(integer)*: number of nonlinear iterations to elapse before next progress message

Number of nonlinear iterations to elapse before next progress message.

(default = 10)

**NLP_ITRLMT** *(integer)*: nonlinear iteration limit

This controls the iteration limit on the number of nonlinear iterations performed.

(default = GAMS IterLim)

**NLP_LINEARZ** *(integer)*: extent to which the solver will attempt to linearize nonlinear models

This determines the extent to which the solver will attempt to linearize nonlinear models.

(default = 0)

- 0 Solver decides
- 1 No linearization occurs
- 2 Linearize ABS MAX and MIN functions
- 3 Same as option 2 plus IF AND OR NOT and all logical operators are linearized

**NLP_MAXLOCALSEARCH** *(integer)*: maximum number of local searches

This controls the maximum number of local searches (multistarts) when solving a NLP using the multistart solver.

(default = 5)

**NLP_MAXLOCALSEARCH_TREE** *(integer)*: maximum number of multistarts

Maximum number of multistarts (at tree nodes)

(default = 1)

**NLP_MAX_RETRY** *(integer)*: maximum number refinement retries to purify the final NLP solution

Maximum number refinement retries to purify the final NLP solution.

(default = 5)

**NLP_MSW_EUCDIST_THRES** *(real)*: euclidean distance threshold in multistart search

Euclidean distance threshold in multistart search

(default = 0.001)

**NLP_MSW_FILTMODE** *(integer)*: filtering mode to exclude certain domains during sampling in multistart search

Filtering mode to exclude certain domains during sampling in multistart search.

(default = -1)

- -1 Solver decides
- +1 Filter-out the points around known KKT or feasible points previously visited
- +2 Filter-out the points whose p are in the vicinity of p(x)
- +4 Filter-out the points in the vicinity of x where x are initial points of all previous local optimizations
- +8 Filter-out the points whose p(.) values are below a dynamic threshold tolerance

**NLP_MSW_MAXNOIMP** *(integer)*: maximum number of consecutive populations to generate without any improvements

Maximum number of consecutive populations to generate without any improvements.

(default = -1)

**NLP_MSW_MAXPOP** *(integer)*: maximum number of populations to generate in multistart search

Maximum number of populations to generate in multistart search.

(default = -1)
**NLP_MSW_MAXREF** (*integer*): maximum number of reference points to generate trial points in multistart search

Maximum number of reference points in the solution space to generate trial points in multistart search.

(default = -1)

**NLP_MSW_NORM** (*integer*): norm to measure the distance between two points in multistart search

Norm to measure the distance between two points in multistart search.

(default = 2)

**NLP_MSW_NUM_THREADS** (*integer*): number of parallel threads to be used when solving an NLP model with the multistart solver

This value specifies the number of parallel threads to be used when solving an NLP model with the multistart solver.

(default = 1)

**NLP_MSW_OVERLAP_RATIO** (*real*): rate of replacement in successive populations

This value specifies the rate of replacement in successive populations. Higher values favors survival of points in the parent population.

(default = 0.1)

**NLP_MSW_POXDIST_THRES** (*real*): penalty function neighborhood threshold in multistart search

Penalty function neighborhood threshold in multistart search

(default = 0.01)

**NLP_MSW_PREPMODE** (*integer*): preprocessing strategies in multistart solver

This value specifies the preprocessing strategies in multistart solver.

(default = -1)

-1 Solver decides
+1 Truncate free variables
+2 Scale reference points to origin
+4 Enable expansive scaling of radius[k] by hit[k]
+8 Skewed sampling allowing values in the vicinity of origin.
+16 Get best bounds by presolver
+32 Get best bounds using GOP
+64 Enable sampling of free variables (not recommended)
+128 Collect sufficiently many trial points prior to local solves
+256 Enable power solver, trying several different local strategies

**NLP_MSW_RG_SEED** (*integer*): random number generator seed for the multistart solver

This value specified the random number generator seed for the multistart solver.

(default = 1019)

**NLP_MSW_SOLIDX** (*integer*): index of the multistart solution to be loaded

Index of the multistart solution to be loaded main solution structures.

(default = 0)

**NLP_MSW_XKKTRAD_FACTOR** (*real*): KKT solution neighborhood factor in multistart search

KKT solution neighborhood factor in multistart search

(default = 0.85)

**NLP_MSW_XNULRAD_FACTOR** (*real*): initial solution neighborhood factor in multistart search
Initial solution neighborhood factor in multistart search
(default = 0.5)

**NLP_PRELEVEL** (*integer*): controls the amount and type of NLP pre-solving

This controls the amount and type of NLP pre-solving.
(default = 126)

+2 Simple pre-solving
+4 Probing
+8 Coefficient reduction
+16 Elimination
+32 Dual reductions
+64 Use dual information
+512 Maximum pass

**NLP_PSTEP_FINITEDIFF** (*real*): value of the step length in computing the derivatives using finite differences

This controls the value of the step length in computing the derivatives using finite differences.
(default = 5e-7)

**NLP_QUADCHK** (*integer*): flag for checking if NLP is quadratic

This is a flag indicating if the nonlinear model should be examined to check if it is a quadratic model.
(default = 1)

0 Do not check if NLP is quadratic
1 Check if NLP is quadratic

**NLP_REDGDTOL** (*real*): tolerance for the gradients of nonlinear functions

This is the tolerance for the gradients of nonlinear functions. The (projected) gradient of a function is considered to be the zero-vector if its norm is below this tolerance.
(default = 1e-7)

**NLP_SOLVER** (*integer*): type of nonlinear solver

This value determines the type of nonlinear solver.
(default = 7)

4 Solver decides
5 Uses Levenberg-Marquardt method to solve nonlinear least-squares problem
6 Uses Barrier solver for convex QCP models
7 Uses CONOPTs reduced gradient solver
8 Uses SLP solver
9 Uses CONOPT with multistart feature enabled

**NLP_SOLVE_AS_LP** (*integer*): flag indicating if the nonlinear model will be solved as an LP

This is a flag indicating if the nonlinear model will be solved as an LP. 1 means that an LP using first order approximations of the nonlinear terms in the model will be used when optimizing the model with the LSOptimize() function.
(default = 0)

0 NLP will not be solved as LP
1 NLP will be solved as LP

**NLP_STALL_ITRLMT** (*integer*): iteration limit before a sequence of non-improving NLP iterations is declared as stalling
This specifies the iteration limit before a sequence of non-improving NLP iterations is declared as stalling, thus causing the solver to terminate.

(default = 100)

**NLP.STARTPOINT** *(integer)*: flag for using initial starting solution for NLP

This is a flag indicating if the nonlinear solver should accept initial starting solutions.

(default = 1)

0 Do not use initial starting solution for NLP

1 Use initial starting solution for NLP

**NLP.SUBSOLVER** *(integer)*: type of nonlinear subsolver

This controls the type of linear solver to be used for solving linear subproblems when solving nonlinear models.

(default = 1)

1 Primal simplex method

2 Dual simplex method

3 Barrier solver with or without crossover

**NLP.USE.CRASH** *(integer)*: flag for using simple crash routines for initial solution

This is a flag indicating if an initial solution will be computed using simple crash routines.

(default = 0)

0 Do not use simple crash routines

1 Use simple crash routines

**NLP.USE.LINDO.CRASH** *(integer)*: flag for using advanced crash routines for initial solution

This is a flag indicating if an initial solution will be computed using advanced crash routines.

(default = 1)

0 Do not use advanced crash routines

1 Use advanced crash routines

**NLP.USE.SELCONEVAL** *(integer)*: flag for using selective constraint evaluations for solving NLP

This is a flag indicating if selective constraint evaluations will be performed in solving a nonlinear model.

(default = 1)

0 Do not use selective constraint evaluations

1 Use selective constraint evaluations

**NLP.USE.SLP** *(integer)*: flag for using sequential linear programming step directions for updating solution

This is a flag indicating if sequential linear programming step directions should be used in updating the solution.

(default = 1)

-1 Solver decides

0 Do not use sequential linear programming step directions

1 Use sequential linear programming step directions

**NLP.USE.STEEPEDGE** *(integer)*: flag for using steepest edge directions for updating solution

This is a flag indicating if steepest edge directions should be used in updating the solution.

(default = 0)

0 Do not use steepest edge directions

1 Use steepest edge directions
**NUM_THREADS** *(integer)*: number of parallel threads to be used

  Synonym: gthreads

  Number of threads to use in the solver routine to be called. It is a solver-independent parameter which internally sets solver-specific threading parameters automatically.

  (default = GAMS Threads)

**PROB_TO_SOLVE** *(integer)*: controls whether the explicit primal or dual form of the given LP problem will be solved

  This flag controls whether the explicit primal or dual form of the given LP problem will be solved.

  (default = 0)

  0 Solver decides

  1 Explicit primal form

  2 Explicit dual form

**PROFILER_LEVEL** *(integer)*: specifies the profiler level to break down the total cpu time into.

  Specifies the profiler level to break down the total cpu time into.

  (default = 0)

  0 Profiler is off

  +1 Enable for simplex solver

  +2 Enable for integer solver

  +4 Enable for multistart solver

  +8 Enable for global solver

**READPARAMS** *(string)*: read Lindo parameter file

**REPORTEVSOL** *(integer)*: solve and report the expected value solution

  (default = 0)

**SAMP_CDSINC** *(real)*: correlation matrix diagonal shift increment

  Correlation matrix diagonal shift increment.

  (default = 1e-6)

**SAMP_NCM_CUTOBJ** *(real)*: objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem

  Objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem.

  (default = 1e-30)

**SAMP_NCM_DSTORAGE** *(integer)*: flag to enable or disable sparse mode in NCM computations

  Flag to enable/disble sparse mode in NCM computations.

  (default = -1)

**SAMP_NCM_ITERLIM** *(integer)*: iteration limit for NCM method

  Iteration limit for NCM method.

  (default = 100)

**SAMP_NCM_METHOD** *(integer)*: bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem

  Bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem.

  (default = 5)

**SAMP_NCM_OPTTOL** *(real)*: optimality tolerance for NCM method

  Optimality tolerance for NCM method.

  (default = 1e-7)
SAMP_SCALE (integer): flag to enable scaling of raw sample data

Flag to enable scaling of raw sample data.
(default = 0)

SOLVER_CONCURRENT_OPTMODE (integer): controls if simplex and interior-point optimizers will run concurrently

Controls if simplex and interior-point optimizers will run concurrently. 0 means no concurrent runs will be performed, 1 means both optimizers will run concurrently if at least two threads exist in system, 2 means both optimizers will run concurrently.
(default = 0)

0 no concurrent runs
1 run concurrently if at least 2 threads exist
2 run concurrently

SOLVER_CUTOFFVAL (real): solver will exit if optimal solution is worse than this

If the optimal objective value of the LP being solved is shown to be worse than this (e.g., if the dual simplex method is being used), then the solver will exit without finding a feasible solution. This is a way of saving computer time if there is no sufficiently attractive solution. SOLVER_USECUTOFFVAL needs to be set to 1 to activate this value.
(default = 0)

SOLVER_FEASTOL (real): feasibility tolerance

This is the feasibility tolerance. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.
(default = 1e-7)

SOLVER_IPMSOL (integer): basis crossover flag for barrier solver

This flag controls whether a basis crossover will be performed when solving LPs with the barrier solver. A value of 0 indicates that a crossover to a basic solution will be performed. If the value is 1, then the barrier solution will be left intact. For example, if alternate optima exist, the barrier method will return a solution that is, loosely speaking, the average of all alternate optima.
(default = 0)

0 Perform crossover to basis solution
1 Leave barrier solution intact

SOLVER_IUSOL (integer): flag for computing basic solution for infeasible model

This is a flag that, when set to 1, will force the solver to compute a basic solution to an infeasible model that minimizes the sum of infeasibilities and a basic feasible solution to an unbounded problem from which an extreme direction originates. When set to the default of 0, the solver will return with an appropriate status flag as soon as infeasibility or unboundedness is detected. If infeasibility or unboundedness is declared with presolver’s determination, no solution will be computed.
(default = 0)

0 Return appropriate status if infeasibility is encountered
1 Force the solver to compute a basic solution to an infeasible model

SOLVER_OPTTOL (real): dual feasibility tolerance

This is the optimality tolerance. It is also referred to as the dual feasibility tolerance. A dual slack (reduced cost) is considered violated if it violates its lower bound by the optimality tolerance.
(default = 1e-7)

SOLVER_PRE_ELIM_FILL (integer): fill-in introduced by the eliminations during pre-solve

This is a nonnegative fill-in value that controls the fill-in introduced by the eliminations during pre-solve. Smaller
values could help when the total nonzeros in the presolved model is significantly more than the original model.
(default = 1000)

**SOLVER_RESTART (integer):** starting basis flag

This is the starting basis flag. 1 means LINDO API will perform warm starts using any basis currently in memory.
0 means LINDO API will perform cold starts discarding any basis in memory and starting from scratch.
(default = 0)

0 Perform cold start
1 Perform warm start

**SOLVER_TIMLMT (integer):** time limit in seconds for continuous solver

This is a time limit in seconds for the LP solver. The default value of -1 imposes no time limit.
(default = GAMS ResLim)

**SOLVER_USECUTOFFVAL (integer):** flag for using cutoff value

This is a flag for the parameter SOLVER_CUTOFFVAL
(default = 0)

0 Do not use cutoff value
1 Use cutoff value

**SPLEX_DPRICING (integer):** pricing option for dual simplex method

This is the pricing option to be used by the dual simplex method.
(default = -1)

-1 Solver decides the dual pricing method
0 Partial pricing
1 Steepest edge

**SPLEX_DUAL_PHASE (integer):** controls the dual simplex strategy

This controls the dual simplex strategy, single-phase versus two-phase.
(default = 0)

0 Solver decides
1 Single-phase
2 Two-phase

**SPLEX_ITRLMT (integer):** simplex iteration limit

This is a limit on the number of iterations the solver will perform before terminating. If this value is a nonnegative integer, then it will be used as an upper bound on the number of iterations the solver will perform. If this value is -1, then no iteration limit will be used. The solution may be infeasible.
(default = infinity)

**SPLEX_PPRICING (integer):** pricing option for primal simplex method

This is the pricing option to be used by the primal simplex method.
(default = -1)

-1 Solver decides the primal pricing method
0 Partial pricing
1 Devex

**SPLEX_REFACFRQ (integer):** number of simplex iterations between two consecutive basis re-factorizations

This is a positive integer scalar referring to the simplex iterations between two consecutive basis re-factorizations.
For numerically unstable models, setting this parameter to smaller values may help.
(default = 100)

**SPLEX_SCALE (integer):** scaling flag

This is the scaling flag. Scaling multiplies the rows and columns of the model by appropriate factors in an attempt to avoid numerical difficulties by reducing the range of coefficient values.

(default = 1)

0 Scaling is suppressed
1 Scaling is performed

**STOC_ABSOPTTOL (real):** absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver

Absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. Possible values are reals in (0,1) interval.

(default = GAMS OptCA)

**STOC_ADD_MPI (integer):** flag to use add-instructions mode when building deteq

Flag to use add-instructions mode when building deteq.

(default = 0)

**STOC_ALD_DUAL_FEASTOL (real):** dual feasibility tolerance for ALD

Dual feasibility tolerance for ALD.

(default = 1e-4)

**STOC_ALD_DUAL_STEPLEN (real):** dual step length for ALD

Dual step length for ALD.

(default = 0.9)

**STOC_ALD_INNER_ITER_LIM (integer):** inner loop iteration limit for ALD

Inner loop iteration limit for ALD.

(default = 1000)

**STOC_ALD_OUTER_ITER_LIM (integer):** outer loop iteration limit for ALD

Outer loop iteration limit for ALD.

(default = 200)

**STOC_ALD_PRIMAL_FEASTOL (real):** primal feasibility tolerance for ALD

Primal feasibility tolerance for ALD.

(default = 1e-4)

**STOC_ALD_PRIMAL_STEPLEN (real):** primal step length for ALD

Primal step length for ALD.

(default = 0.5)

**STOC_AUTOAGGR (integer):** flag to enable or disable autoaggregation

Flag to enable or disable autoaggregation.

(default = 1)

**STOC_BENCHMARK_SCEN (integer):** benchmark scenario to compare EVPI and EVMU against

Benchmark scenario to compare EVPI and EVMU against.

(default = -2)

**STOC_BIGM (real):** big-M value for linearization and penalty functions
Big-M value for linearization and penalty functions.
(default = 1e7)

**STOC_BUCKET_SIZE (integer):** bucket size in Benders decomposition

Bucket size in Benders decomposition. Possible values are positive integers or (-1) for solver decides.
(default = -1)

**STOC_CALC_EVPI (integer):** flag to enable or disable calculation of EVPI

Flag to enable/disable calculation of lower bounds on EVPI.
(default = 1)

0 disable
1 enable

**STOC_CORRELATION_TYPE (integer):** correlation type associated with correlation matrix

Correlation type associated with the correlation matrix.
(default = 0)

-1 Target correlation
0 Pearson correlation
1 Kendall correlation
2 Spearman correlation

**STOC_DEQOPT (integer):** method to solve the DETEQ problem

Optimization method to solve the DETEQ problem.
(default = 0)

0 Solver decides
1 Use primal method
2 Use dual simplex
3 Use barrier solver
4 Use NLP solver
6 Use multi-start solver
7 Use global solver

**STOC_DETEQ_TYPE (integer):** type of deterministic equivalent

Type of deterministic equivalent to be used by the solver. Implicit deterministic equivalent is valid for linear and integer models only.
(default = -1)

-1 Solver decides
0 Implicit deterministic equivalent
1 Explicit deterministic equivalent

**STOC_DS_SUBFORM (integer):** subproblem formulation to use in DirectSearch

This parameter specifies the type of subproblem formulation to be used in heuristic search.
(default = -1)

-1 Solver decides
0 Perform heuristic search in the original solution space
Perform heuristic search in the space of discrete variables coupled with optimizations in the linear space.

**STOC_ELIM_FXVAR (integer):** flag to enable elimination of fixed variables from deteq MPI

Flag to enable elimination of fixed variables from deteq MPI.

(default = 1)

**STOC_INFBND (real):** value to truncate infinite bounds at non-leaf nodes

Value to truncate infinite bounds at nonleaf nodes.

(default = 1e9)

**STOC_ITER_LIM (integer):** iteration limit for stochastic solver

Iteration limit for stochastic solver. Possible values are positive integers or (-1) no limit.

(default = infinity)

**STOC_MAP_MPI2LP (integer):** flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements

Flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements.

(default = 0)

**STOC_MAX_NUMSCENS (integer):** maximum number of scenarios before forcing automatic sampling

Maximum number of scenarios before forcing automatic sampling. Possible values are positive integers.

(default = 40000)

**STOC_METHOD (integer):** stochastic optimization method to solve the model

Stochastic optimization method to solve the model.

(default = -1)

-1 Solve with the method chosen by the solver

0 Solve the deterministic equivalent (DETEQ)

1 Solve with the Nested Benders Decomposition (NBD) method

**STOC_NAMEDATA_LEVEL (integer):** name data level

Name data level.

(default = 1)

**STOC_NODELP_PRELEVEL (integer):** presolve level solving node-models

Presolve level solving node-models.

(default = 0)

+2 Simple pre-solving
+4 Probing
+8 Coefficient reduction
+16 Elimination
+32 Dual reductions
+64 Use dual information
+512 Maximum pass

**STOC_NSAMPLE_PER_STAGE (string):** list of sample sizes per stage (starting at stage 2)

Comma separated list of sample sizes per stage. The sample size of stage 1 is assumed to be 1 so that this list starts with stage 2.

**STOC_NSAMPLE_SPAR (integer):** common sample size per stochastic parameter
Common sample size per stochastic parameter. Possible values are positive integers.
(default = -1)

**STOC_NSAMPLE_STAGE** (integer): common sample size per stage

Common sample size per stage.
(default = -1)

**STOC_NUM_THREADS** (integer): number of parallel threads

This value specifies the number of parallel threads to be used when solving a stochastic programming model.
(default = 1)

**STOC_RELOPTTOL** (real): relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver

Relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. Possible values are reals in (0,1) interval.
(default = GAMS OptCR)

**STOC_REL_DSTEPTOL** (real): dual-step tolerance

This value specifies the dual-step tolerance in decomposition based algorithms.
(default = 1e-7)

**STOC_REL_PSTEPTOL** (real): primal-step tolerance

This value specifies the primal-step tolerance in decomposition based algorithms.
(default = 1e-8)

**STOC_REOPT** (integer): reoptimization method to solve the node-models

Reoptimization method to solve the node-models.
(default = 0)

0 Solver decides
1 Use primal method
2 Use dual simplex
3 Use barrier solver
4 Use NLP solver

**STOC_RG_SEED** (integer): seed to initialize the random number generator

Seed to initialize the random number generator. Possible values are positive integers.
(default = 1031)

**STOC_SAMP_CONT_ONLY** (integer): flag to restrict sampling to continuous stochastic parameters only or not

Flag to restrict sampling to continuous stochastic parameters only or not.
(default = 1)

0 disable
1 enable

**STOC_SBD_MAXCUTS** (integer): max cuts to generate for master problem

Max cuts to generate for master problem.
(default = -1)

**STOC_SBD_NUMCANDID** (integer): maximum number of candidate solutions to generate at SBD root

Maximum number of candidate solutions to generate at SBD root.
(default = -1)

**STOC_SBD_OBJCUTFLAG (integer)**: flag to enable objective cut in SBD master problem

Flag to enable objective cut in SBD master problem.

(default = 1)

**STOC_SBD_OBJCUTVAL (real)**: RHS value of objective cut in SBD master problem

RHS value of objective cut in SBD master problem.

(default = 1e-30)

**STOC_SHARE_BEGSTAGE (integer)**: stage beyond which node-models are shared

Stage beyond which node-models share the same model structure. Possible values are positive integers less than or equal to number of stages in the model or (-1) for solver decides.

(default = -1)

**STOC_TIME_LIM (real)**: time limit for stochastic solver

Time limit for stochastic solver. Possible values are nonnegative real numbers or -1 for solver decides.

(default = GAMS ResLim)

**STOC_TOPOPT (integer)**: optimization method to solve the root problem

Optimization method to solve the root problem.

(default = 0)

0 Solver decides
1 Use primal method
2 Use dual simplex
3 Use barrier solver
4 Use NLP solver
5 Use multi-start solver
6 Use global solver
7 Use multi-start solver
8 Use global solver

**STOC_VARCONTROL_METHOD (integer)**: sampling method for variance reduction

Sampling method for variance reduction.

(default = 1)

0 Montecarlo sampling
1 Latsinsquare sampling
2 Antithetic sampling

**STOC_WSBAS (integer)**: warm start basis for wait-see model

Warm start basis for wait-see model.

(default = -1)

**SVR_LS_ANTITHETIC (string)**: Sample variance reduction map to Lindo Antithetic algorithm

**SVR_LS_LATINSQUARE (string)**: Sample variance reduction map to Lindo Latin Square algorithm

**SVR_LS_MONTECARLO (string)**: Sample variance reduction map to Lindo Montecarlo algorithm

**USEGOP (integer)**: use global optimization

This value determines whether the global optimization will be used.

(default = 1)

0 Do not use global optimization
1 Use global optimization

**WRITEDEMPI (string):** write deterministic equivalent in MPI format

**WRITEDEMPS (string):** write deterministic equivalent in MPS format

**WRITEMPI (string):** write (S)MPI file of processed model

If this option is set, Lindo write an MPI file of processed model. If set, the value of this option defines the name of the MPI file.

**WRITEMPS (string):** write (S)MPS file of processed model

### 7 Stochastic Programming (SP) in GAMS/Lindo

GAMS/Lindo can also solve stochastic programming models. The syntax to set up an SP problem in GAMS is explained in the chapter "Stochastic Programming (SP) with EMP". The options to control LINDOs stochastic solver are described in the subsection SP Options.
LocalSolver

www.localsolver.com

LocalSolver is an innovative mathematical programming solver developed and maintained by the company Innovation 24. LocalSolver finds high-quality solutions for large-scale mixed-variable non-convex optimization problems. Based on a unique hybrid neighborhood search approach, it combines local search techniques, constraint propagation and inference techniques, linear and mixed-integer programming techniques, as well as nonlinear programming techniques.

LocalSolver works on the algebraic representation of the model. As such, most common GAMS functions are supported, including logical operators and equations. However, GAMS extrinsic functions cannot be used together with LocalSolver. Also special-ordered-set variables are currently not supported.

LocalSolver requires all variables to have finite bounds. Currently, the GAMS/LocalSolver link sets missing variable bounds to ±1e10, but the user is invited to provide proper bounds. However, no bounds should be set for variables that can be aggregate out by the solver link (aggrvars option).

LocalSolver is usually not able to prove infeasibility of the problem or global optimality of a solution and thus continues the search until interrupted by a limit (iterlim, reslim) or the user (Ctrl+C) or when found a solution with value equal or better of the dualbound option. It is thus important to set an iteration limit or a time limit that is appropriate to the model.

1 Using GAMS/LocalSolver

The following statement can be used inside your GAMS program to specify using LocalSolver:

```
Option MIP = LocalSolver; { or MIQCP or MINLP or ... }
```

The above statement should appear before the solve statement. If LocalSolver was specified as the default solver during GAMS installation, the above statement is not necessary.

The following GAMS options are used by GAMS/LocalSolver: iterlim, reslim, and threads.

Additionally, a GAMS/LocalSolver run can be customized by a using a solver options file. With the following example, one disables the automated aggregation of variables in the GAMS/LocalSolver link and sets the seed for the random number generator in LocalSolver to 42:

```
aggrvars 0
seed 42
```

The following options can be set:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggrvars</td>
<td>Automatically find and aggregate out variables. If this option is enabled, the GAMS/LocalSolver link will look for free continuous variables that it can substitute out by using equality equations in which these variables appear linearly. A heuristic procedure has been implemented to substitute out as many variables as possible.</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>annealinglevel</td>
<td>Simulated annealing level (higher numbers increase the number of uphill moves, thus increases chances to reach better solutions).</td>
<td>1</td>
</tr>
<tr>
<td>dualbound</td>
<td>Dual bound on objective function (lower bound for minimization, upper bound for maximization). If a value has been set, GAMS/LocalSolver will stop when it found a feasible solution with an objective function value equal or better to the dual bound.</td>
<td>unused</td>
</tr>
<tr>
<td>origlog</td>
<td>Whether to show original LocalSolver log instead of GAMS/LocalSolver log. Note, that LocalSolver prints its log always to stdout, despite of the value of the GAMS lo option.</td>
<td>0</td>
</tr>
<tr>
<td>seed</td>
<td>Seed of random number generator.</td>
<td>0</td>
</tr>
<tr>
<td>solvtrace</td>
<td>Filename of solving trace file.</td>
<td>0</td>
</tr>
<tr>
<td>solvtracetime</td>
<td>Time interval when a trace record is written.</td>
<td>1</td>
</tr>
<tr>
<td>writelnsp</td>
<td>Name of file into which to write instance as LocalSolver .lsp file.</td>
<td></td>
</tr>
</tbody>
</table>

2 Using a LocalSolver library license

Users that have a GAMS/LocalSolver link license need to provide a separate LocalSolver license file license.dat. By default, LocalSolver looks for its license file in 3 different locations in the following order:

1. In the file specified by the environment variable LS_LICENSE_PATH.
2. In the current directory, that is the directory from where GAMS is spawned. If a file license.dat exists in this directory, it is used as default license.
3. In /opt/localsolver_X_Y/license.dat on Linux and MacOS X and C:\localsolver_X_Y\license.dat on Windows.
# Contents

1 Introduction .......................................................... 324
2 How to Run a Model with GAMS/MINOS .......................... 324
3 Overview of GAMS/MINOS ........................................... 325
   3.1 Linear Programming ............................................. 325
   3.2 Problems with a Nonlinear Objective ......................... 326
   3.3 Problems with Nonlinear Constraints ......................... 327
4 Modeling Issues ....................................................... 328
   4.1 Starting Points .................................................. 328
   4.2 Bounds .......................................................... 329
   4.3 Scaling .......................................................... 329
   4.4 The Objective Function ........................................ 329
5 GAMS Options ......................................................... 330
   5.1 Options Specified through the Option Statement .......... 330
   5.2 Options Specified through Model Suffixes ................ 332
6 Summary of MINOS Options ........................................... 333
   6.1 Output related options ......................................... 333
   6.2 Tolerances ...................................................... 333
   6.3 Limits ........................................................... 333
   6.4 Other algorithmic options ..................................... 334
   6.5 Examples of GAMS/MINOS Option File ....................... 334
7 Special Notes .......................................................... 335
   7.1 Modeling Hints .................................................. 335
   7.2 Storage .......................................................... 335
8 The GAMS/MINOS Log File .......................................... 336
   8.1 Linear Programs ................................................ 336
   8.2 Linearly Constrained NLP’s ................................... 337
   8.3 NLP’s with Nonlinear Constraints ............................ 338
9 Detailed Description of MINOS Options ........................... 340
10 Exit Conditions ....................................................... 350

Bibliography .......................................................... 352
1 Introduction

This document describes the GAMS interface to MINOS which is a general purpose nonlinear programming solver. GAMS/MINOS is a specially adapted version of the solver that is used for solving linear and nonlinear programming problems in a GAMS environment.

GAMS/MINOS is designed to find solutions that are locally optimal. The nonlinear functions in a problem must be smooth (i.e., their first derivatives must exist). The functions need not be separable. Integer restrictions cannot be imposed directly.

A certain region is defined by the linear constraints in a problem and by the bounds on the variables. If the nonlinear objective and constraint functions are convex within this region, any optimal solution obtained will be a global optimum. Otherwise there may be several local optima, and some of these may not be global. In such cases the chances of finding a global optimum are usually increased by choosing a staring point that is "sufficiently close", but there is no general procedure for determining what "close" means, or for verifying that a given local optimum is indeed global.

Linearly constrained models are solved with a very efficient and reliable reduced gradient technique, that utilizes the sparsity of the model. Models with nonlinear constraints are solved with a method that iteratively solves subproblems with linearized constraints and an augmented Lagrangian objective function. This iterative scheme implies that only the final optimal solution is feasible for nonlinear models, in contrast to the feasible path method used by some other NLP solvers, e.g., CONOPT. MINOS and CONOPT are very complementary to each other as they employ very different algorithms. See MINOS vs CONOPT for a comparison of both.

GAMS allows you to specify values for many parameters that control GAMS/MINOS, and with careful experimentation you may be able to influence the solution process in a helpful way. All MINOS options available through GAMS/MINOS are summarized at the end of this document.

2 How to Run a Model with GAMS/MINOS

MINOS is capable of solving models of the following types: LP, NLP, DNLP and RMINLP. If MINOS is not specified as the default LP, NLP, DNLP or RMINLP solver, then the following statement can be used in your GAMS model:

```plaintext
option nlp=minos; { or lp or dnlp or rminlp }
```

or

```plaintext
option nlp=minos55; { or lp or dnlp or rminlp }
```

It should appear before the ```solve``` statement.

This will invoke MINOS 5.5. In some cases an older version of MINOS, version 5.4 is more efficient than the newer version. MINOS 5.4 can be selected by:

```plaintext
option nlp=minos5; { or lp or dnlp or rminlp }
```

To be complete, we mention that this can be also specified on the command line, as in:

```plaintext
> gams camcge nlp=minos
```

This will override the global default, but if an algorithm option has been specified inside the model, then that specification takes precedence.
3 Overview of GAMS/MINOS

GAMS/MINOS is a system designed to solve large-scale optimization problems expressed in the following form:

\[
\begin{align*}
\text{NLP} & \quad \text{minimize} \quad F(x) + c^T x + d^T y \\
& \quad \text{subject to} \quad f(x) + A_1 y \sim b_1 \\
& \quad \quad A_2 x + A_3 y \sim b_2 \\
& \quad \quad \ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u
\end{align*}
\]

where the vectors \( c, d, b_1, b_2, \ell, u \) and the matrices \( A_1, A_2, A_3 \) are constant, \( F(x) \) is a smooth scalar function, and \( f(x) \) is a vector of smooth functions. The \( \sim \) signs mean that individual constraints may be defined using \( \leq, = \) or \( \geq \) corresponding to the GAMS constructs =L=, =E= and =G=.

The components of \( x \) are called the nonlinear variables, and the components of \( y \) are the linear variables. Similarly, the equations in (2) are called the nonlinear constraints, and the equations in (3) are the linear constraints. Equations (2) and (3) together are called the general constraints.

Let \( m_1 \) and \( n_1 \) denote the number of nonlinear constraints and variables, and let \( m \) and \( n \) denote the total number of (general) constraints and variables. Thus, \( A_3 \) has \( m - m_1 \) rows and \( n - n_1 \) columns. The constraints (4) specify upper and lower bounds on all variables. These are fundamental to many problem formulations and are treated specially by the solution algorithms in GAMS/MINOS. Some of the components of \( \ell \) and \( u \) may be \( -\infty \) or \( +\infty \) respectively, in accordance with the GAMS use of \(-\text{INF}\) and \(+\text{INF}\).

The vectors \( b_1 \) and \( b_2 \) are called the right-hand side, and together are denoted by \( b \).

3.1 Linear Programming

If the functions \( F(x) \) and \( f(x) \) are absent, the problem becomes a linear program. Since there is no need to distinguish between linear and nonlinear variables, we use \( x \) rather than \( y \). GAMS/MINOS converts all general constraints into equalities, and the only remaining inequalities are simple bounds on the variables. Thus, we write linear programs in the form

\[
\begin{align*}
\text{LP} & \quad \text{minimize} \quad c^T x \\
& \quad \text{subject to} \quad Ax + Is = 0 \\
& \quad \quad \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u
\end{align*}
\]

where the elements of \( x \) are your own GAMS variables, and \( s \) is a set of slack variables: one for each general constraint. For computational reasons, the right-hand side \$b\$ is incorporated into the bounds on \( s \).

In the expression \( Ax + Is = 0 \) we write the identity matrix explicitly if we are concerned with columns of the associated matrix \((A \ I)\). Otherwise we will use the equivalent notation \( Ax + s = 0 \).

GAMS/MINOS solves linear programs using a reliable implementation of the primal simplex method \([3]\), in which the constraints \( Ax + Is = 0 \) are partitioned into the form

\[ Bx_B + Nx_N = 0, \]

where the basis matrix is square and nonsingular. The elements of \( x_B \) and \( x_N \) are called the basic or nonbasic variables respectively. Together they are a permutation of the vector

\[ \begin{pmatrix} x \\ s \end{pmatrix}. \]

Normally, each nonbasic variable is equal to one of its bounds, and the basic variables take on whatever values are needed to satisfy the general constraints. (The basic variables may be computed by solving the linear equations \( Bx_B = Nx_N \).) It can be shown that if an optimal solution to a linear program exists, then it has this form.
The simplex method reaches such a solution by performing a sequence of iterations, in which one column of \( B \) is replaced by one column of \( N \) (and vice versa), until no such interchange can be found that will reduce the value of \( c^T x \).

As indicated nonbasic variables usually satisfy their upper and lower bounds. If any components of \( x_B \) lie significantly outside their bounds, we say that the current point is infeasible. In this case, the simplex method uses a Phase 1 procedure to reduce the sum of infeasibilities to zero. This is similar to the subsequent Phase 2 procedure that optimizes the true objective function \( c^T x \).

If the solution procedures are interrupted, some of the nonbasic variables may lie strictly between their bounds \( \ell_j < x_j < u_j \). In addition, at a “feasible” or “optimal” solution, some of the basic variables may lie slightly outside their bounds: \( \ell_j - \delta < x_j < u_j < u_j + \delta \) where \( \delta \) is a feasibility tolerance (typically \( 10^{-6} \)). In rare cases, even nonbasic variables might lie outside their bounds by as much as \( \delta \).

GAMS/MINOS maintains a sparse \( LU \) factorization of the basis matrix \( B \), using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the Fortran package LUSOL \([7]\) (see \([1, 2, 8, 9]\)). The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

### 3.2 Problems with a Nonlinear Objective

When nonlinearities are confined to the term \( F(x) \) in the objective function, the problem is a linearly constrained nonlinear program. GAMS/MINOS solves such problems using a reduced-gradient algorithm \([11]\) combined with a quasi-Newton algorithm that is described in \([4]\). In the reduced-gradient method, the constraints \( Ax + Is = 0 \) are partitioned into the form

\[
Bx_B + Sx_S + Nx_N = 0
\]

where \( x_s \) is a set of superbasic variables. At a solution, the basic and superbasic variables will lie somewhere between their bounds (to within the feasibility tolerance \( \delta \), while nonbasic variables will normally be equal to one of their bounds, as before. Let the number of superbasic variables be \( s \), the number of columns in \( S \). (The context will always distinguish $s$ from the vector of slack variables.) At a solution, \( s \) will be no more than \( n_1 \), the number of nonlinear variables. In many practical cases we have found that \( s \) remains reasonably small, say 200 or less, even if \( n_1 \) is large.

In the reduced-gradient algorithm, \( x_j \) is regarded as a set of “independent variables” or “free variables” that are allowed to move in any desirable direction, namely one that will improve the value of the objective function (or reduce the sum of infeasibilities). The basic variables can then be adjusted in order to continue satisfying the linear constraints.

If it appears that no improvement can be made with the current definition of \( B, S \) and \( N \), some of the nonbasic variables are selected to be added to \( S \), and the process is repeated with an increased value of \( s \). At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of \( s \) is reduced by one.

A step of the reduced-gradient method is called a minor iteration. For linear problems, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variable oscillating between 0 and 1.

A certain matrix \( Z \) is needed now for descriptive purposes. It takes the form

\[
\begin{pmatrix}
-B^{-1}S \\
I \\
0
\end{pmatrix}
\]

though it is never computed explicitly. Given an \( LU \) factorization of the basis matrix \( B \), it is possible to compute products of the form \( Zq \) and \( Z^T g \) by solving linear equations involving \( B \) or \( B^T \). This in turn allows optimization to be performed on the superbasic variables, while the basic variables are adjusted to satisfy the general linear constraints.

An important feature of GAMS/MINOS is a stable implementation of a quasi-Newton algorithm for optimizing the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the \( B, S, N \) partition remains constant. A search direction \( q \) for the superbasic variables is obtained by solving a system of the form

\[
R^T R q = -Z^T g
\]

where \( g \) is a gradient of \( F(x) \), \( Z^T g \) is the reduced gradient, and \( R \) is a dense upper triangular matrix. GAMS computes the gradient vector \( g \) analytically, using symbolic differentiation. The matrix \( R \) is updated in various ways in order to approximate the reduced Hessian according to \( R^T R \approx Z^T H Z \) where \( H \) is the matrix of second derivatives of \( F(x) \) (the Hessian).
Once \( q \) is available, the search direction for all variables is defined by \( p = Zq \). A ine search is then performed to find an approximate solution to the one-dimensional (w.r.t. \( \alpha \) ) problem

\[
\begin{align*}
\text{minimize } & \quad F(x + \alpha p) \\
\text{subject to } & \quad 0 < \alpha < \beta
\end{align*}
\]

where \( \beta \) is determined by the bounds on the variables. Another important piece in GAMS/MINOS is a step-length procedure used in the linesearch to determine the step-length \( \alpha \) (see [6]). The number of nonlinear function evaluations required may be influenced by setting the Linesearch tolerance, as discussed in Section Detailed Description of MINOS Options.

As a linear programming solver, an equation \( B^T \pi = gB \) is solved to obtain the dual variables or shadow prices \( \pi \) where \( gB \) is the gradient of the objective function associated with basic variables. It follows that \( gB - B^T \pi = 0 \). The analogous quantity for superbasic variables is the reduced-gradient vector \( Z^T g = g_x - s^T \pi \); this should also be zero at an optimal solution. (In practice its components will be of order \( r ||\pi|| \) where \( r \) is the optimality tolerance, typically \( 10^{-6} \), and \( ||\pi|| \) is a measure of the size of the elements of \( \pi \).)

### 3.3 Problems with Nonlinear Constraints

If any of the constraints are nonlinear, GAMS/MINOS employs a project Lagrangian algorithm, based on a method due to [10], see [5]. This involves a sequence of major iterations, each of which requires the solution of a linearly constrained subproblem. Each subproblem contains linearized versions of the nonlinear constraints, as well as the original linear constraints and bounds.

At the start of the \( k \)th major iteration, let \( x_k \) be an estimate of the nonlinear variables, and let \( \lambda_k \) be an estimate of the Lagrange multipliers (or dual variables) associated with the nonlinear constraints. The constraints are linearized by changing \( f(x) \) in equation (2) to its linear approximation:

\[
f'(x, x_k) = f(x_k) + J(x_k)(x - x_k)
\]

or more briefly

\[
f' = f_k + J_k(x - x_k)
\]

where \( J(x_k) \) is the Jacobian matrix evaluated at \( x_k \). (The \( i \)th row of the Jacobian is the gradient vector of the \( i \)th nonlinear constraint function. As for the objective gradient, GAMS calculates the Jacobian using symbolic differentiation).

The subproblem to be solved during the \( k \)th major iteration is then

\[
\begin{align*}
\text{minimize } & \quad F(x) + \epsilon x^T x + \delta^T y - \lambda_k^T (f - f') + 0.5 \rho (f - f')^T (f - f') \\
\text{subject to } & \quad f' + A_1 y \sim b_1 \\
& \quad A_2 x + A_3 y \sim b_2 \\
& \quad \ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u
\end{align*}
\]

The objective function (5) is called an augmented Lagrangian. The scalar \( \rho \) is a penalty parameter, and the term involving \( \rho \) is a modified quadratic penalty function.

GAMS/MINOS uses the reduced-gradient algorithm to minimize (5) subject to (6) – (8). As before, slack variables are introduced and \( b_1 \) and \( b_2 \) are incorporated into the bounds on the slacks. The linearized constraints take the form

\[
\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}
\]

This system will be referred to as \( Ax + Is = 0 \) as in the linear case. The Jacobian \( J_k \) is treated as a sparse matrix, the same as the matrices \( A_1, A_2, \) and \( A_3 \).

In the output from GAMS/MINOS, the term Feasible subproblem indicates that the linearized constraints have been satisfied. In general, the nonlinear constraints are satisfied only in the limit, so that feasibility and optimality occur at essentially the same time. The nonlinear constraint violation is printed every major iteration. Even if it is zero early on (say at the initial point), it may increase and perhaps fluctuate before tending to zero. On “well behaved problems”, the constraint violation will decrease quadratically (i.e., very quickly) during the final few major iteration.
4 Modeling Issues

Formulating nonlinear models requires that the modeler pays attention to some details that play no role when dealing with linear models.

4.1 Starting Points

The first issue is specifying a starting point. It is advised to specify a good starting point for as many nonlinear variables as possible. The GAMS default of zero is often a very poor choice, making this even more important.

As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points \((x_i, y_i)\):

<table>
<thead>
<tr>
<th>Example</th>
<th>minimize (r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject to ((x_i-a)^2 + (y_i-b)^2 \leq r^2), (r \geq 0).</td>
<td></td>
</tr>
</tbody>
</table>

This problem can be modeled in GAMS as follows.

```gams
set i 'points' /p1*p10/;
parameters
   x(i) 'x coordinates',
   y(i) 'y coordinates';
* fill with random data
   x(i) = uniform(1,10);
   y(i) = uniform(1,10);
variables
   a 'x coordinate of center of circle'
   b 'y coordinate of center of circle'
   r 'radius';
equations
   e(i) 'points must be inside circle';
e(i).
   sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);
   r.lo = 0;
model m /all/;
option nlp=minos;
solve m using nlp minimizing r;
```

Without help, MINOS will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

\[
\begin{align*}
    x_{\min} &= \min_i x_i, \\
    y_{\min} &= \min_i y_i, \\
    x_{\max} &= \max_i x_i, \\
    y_{\max} &= \max_i y_i,
\end{align*}
\]
then good estimates are

\[ a = \frac{x_{\min} + x_{\max}}{2}, \]
\[ b = \frac{y_{\min} + y_{\max}}{2}, \]
\[ r = \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \]

Thus we include in our model:

```gams
parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, x(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );
```

and now the model solves very easily.

Level values can also be set implicitly as a result of assigning bounds. When a variable is bounded away from zero, for instance by the statement `Y.LO = 1;`, the `SOLVE` statement will override the default level of zero of such a variable in order to make it feasible.

### 4.2 Bounds

Setting appropriate bounds can be very important to guide the algorithm from visiting uninteresting areas, and to prevent function evaluation errors to happen.

If your model contains an expression of the form \( x^y \) it is important to add a bound \( x > 0.001 \), as exponentation is evaluated in GAMS as \( \exp(y \log(x)) \). In some cases one cannot write a bound directly, e.g. if the equation is \( z = x^y \). In that case it is advised to introduce an extra variable and equation:

\[ z = x^\vartheta \]
\[ \vartheta = f(y) \]
\[ \vartheta \geq \epsilon \]

(Note that the function \( SQR(x) \) does not require \( x \) to be positive).

If the model produces function evaluation errors adding bounds is preferred to raising the `DOMLIM` limit.

Bounds in GAMS are specified using `X.LO(i)=0.001` and `X.UP(i) = 1000`.

### 4.3 Scaling

Although MINOS has some facilities to scale the problem before starting to optimize it, it remains in important task for the modeler to provide a well-scaled model. This is especially the case for nonlinear models. GAMS has special syntax features to specify row and column scales that allows the modeler to keep the equations in a most natural form. For more information consult the GAMS User’s Guide.

### 4.4 The Objective Function

The first step GAMS/MINOS performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. MINOS however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:
obj.. z =e= sum(i, sqr(resid(i)));  

model m /all/;  
solve m using nlp minimizing z;  

This can be cast in form NLP (equations (1) – (4)) by saying minimize $z$ subject to $z = \sum_i resid_i^2$ and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize $\sum_i resid_i^2$ directly. This can be achieved by a simple reformulation: $z$ can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable $z$ is a free continuous variable (no bounds are defined on $z$),
- $z$ appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- $z$ is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by MINOS. For instance the model chem.gms from the model library solves in 21 iterations. When we add the bound

energy.lo = 0;  

on the objective variable energy and thus preventing it from being substituted out, MINOS will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

\[
\begin{align*}
\text{Variables} & \quad x, y, z; \\
\text{Equations} & \quad \text{e1, e2;} \\
\text{e1..} z & =e= y; \\
\text{e2..} y & =e= \text{sqr(1+x)}; \\
\text{model m /all/;} \\
\text{option nlp=mins;} \\
\text{solve m using nlp minimizing z;} \\
\end{align*}
\]

will be reformulated as an unconstrained optimization problem

\[
\text{minimize } f(x) = (1+x)^2.
\]

These additional reformulations can be turned off by using the statement option reform = 0; (see Section GAMS Options).

## 5 GAMS Options

The following GAMS options are used by GAMS/MINOS:

### 5.1 Options Specified through the Option Statement

The following options are specified through the option statement. For example,

option iterlim = 100 ;
sets the iteration limit to 100.

**LP**

This option selects the LP solver. Example: `option LP=MINOS;` See also Section How to Run a Model with GAMS/MINOS.

**NLP**

This option selects the NLP solver. Example: `option NLP=MINOS;` See also Section How to Run a Model with GAMS/MINOS.

**DNLP**

Selects the DNLP solver for models with discontinuous or non-differentiable functions. Example: `option DNLP=MINOS;` See also Section How to Run a Model with GAMS/MINOS.

**RMIP**

Selects the Relaxed Mixed-Integer (RMIP) solver. By relaxing the integer conditions of a MIP model, effectively an LP model results. Example: `option RMIP=MINOS;` See also Section How to Run a Model with GAMS/MINOS.

**RMINLP**

Selects the Relaxed Non-linear Mixed-Integer (RMINLP) solver. By relaxing the integer conditions in an MINLP, the model becomes effectively an NLP. Example: `option RMINLP=MINOS;` See also Section How to Run a Model with GAMS/MINOS.

**iterlim**

Sets the (minor) iteration limit. Example: `option iterlim=50000;` The default is 10000. MINOS will stop as soon as the number of minor iterations exceeds the iteration limit. In that case the current solution will be reported.

**reslim**

Sets the time limit or resource limit. Depending on the architecture this is wall clock time or CPU time. MINOS will stop as soon as more than reslim seconds have elapsed since MINOS started. The current solution will be reported in this case. Example: `option reslim = 600;` The default is 1000 seconds.

**domlim**

Sets the domain violation limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating $x^y$ for $x < \epsilon$ ($x^y$ is evaluated as $\exp(y \log x)$). When such a situation occurs the number of domain errors is increased by one, and MINOS will stop if this number exceeds the limit. If the limit has not been reached, a reasonable number is returned (e.g., in the case of $\sqrt{x} < 0$ a zero is passed back) and MINOS is asked to continue. In many cases MINOS will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don’t occur. For example, when an expression $\log(x)$ is present in the model, add a statement like $x.lo = 0.001;$. Example: `option domlim=100;` The default value is 0.

**bratio**

Basis acceptance test. When several models are solved in a row, GAMS automatically passes dual information to MINOS so that it can reconstruct an advanced basis. When too many new variables or constraints enter the model, it may be better not to use existing basis information, but to crash a new basis instead. The bratio determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis. Example: `option bratio=1.0;` Default: bratio = 0.25.

**sysout**

Debug listing. When turned on, extra information printed by MINOS will be added to the listing file. Example: `option sysout=on;` Default: sysout = off.

**work**

The work option sets the amount of memory MINOS can use. By default an estimate is used based on the model
statistics (number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes etc.). In most cases this is sufficient to solve the model. In some extreme cases MINOS may need more memory, and the user can specify this with this option. For historical reasons work is specified in "double words" or 8 byte quantities. For example, option work=100000; will ask for 0.76 MB (a megabyte being defined as 1024 × 1024 bytes).

**reform**

This option will instruct the reformulation mechanism described in Section The Objective Function to substitute out equality equations. The default value of 100 will cause the procedure to try further substitutions along the diagonal after the objective variable has been removed. Any other value will prohibit this diagonal procedure. Example: option reform = 0; Default: reform = 100.

### 5.2 Options Specified through Model Suffixes

The following options are specified through the use of the model suffix. For example:

```gams
model m /all/;
m.workspace = 10;
solve m using nlp minimizing z;
```

sets the amount of memory used to 10 MB. "m" is the name of the model as specified by the model statement. In order to be effective, the assignment of the model suffix should be made between the model and solve statements.

**m.iterlim**

Sets the iteration limit. Overrides the global iteration limit. Example: m.iterlim=50000; The default is 10000. See also Section GAMS Options.

**m.reslim**

Sets the resource or time limit. Overrides the global resource limit. Example: m.reslim=600; The default is 1000 seconds. See also Section GAMS Options.

**m.bratio**

Sets the basis acceptance test parameter. Overrides the global setting. Example: m.bratio=1.0; The default is 0.25. See also Section GAMS Options.

**m.scaleopt**

Whether or not to scale the model using user-supplied scale factors. The user can provide scale factors using the scale variable and equation suffix. For example, x.scale(i,j) = 100; will assign a scale factor of 100 to all x_{i,j} variables. The variables MINOS will see are scaled by a factor 1/variable_scale, so the modeler should use scale factors that represent the order of magnitude of the variable. In that case MINOS will see variables that are scaled around 1.0. Similarly equation scales can be assigned to equations, which are scaled by a factor 1/equation_scale. Example: m.scaleopt=1; will turn scaling on. The default is not to use scaling, and the default scale factors are 1.0. Automatic scaling is provided by the MINOS option scale option.

**m.optfile**

Sets whether or not to use a solver option file. Solver specific MINOS options are specified in a file called minos.opt, see Section Detailed Description of MINOS Options. To tell MINOS to use this file, add the statement: option m.optfile=1; The default is not to use an option file.

**m.workspace**

The workspace option sets the amount of memory that MINOS can use. By default an estimate is used based on the model statistics (number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes, etc.). In most cases this is sufficient to solve the model. In some extreme cases MINOS may need more memory, and the user can specify this with this option. The amount of memory is specified in MB. Example: m.workspace = 5;
6 Summary of MINOS Options

The performance of GAMS/MINOS is controlled by a number of parameters or "options." Each option has a default value that should be appropriate for most problems. For special situations it is possible to specify non-standard values for some or all of the options through the MINOS option file.

All these options should be entered in the option file minos.opt (for the older solver MINOS5 this name is 'minos5.opt' after setting the m.OPTFILE parameter to 1. The option file is not case sensitive and the keywords must be given in full. Examples for using the option file can be found at the end of this section. The second column in the tables below contains the section where more detailed information can be obtained about the corresponding option in the first column.

6.1 Output related options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug level</td>
<td>Controls amount of debug information written</td>
<td>0</td>
</tr>
<tr>
<td>log frequency</td>
<td>Number of iterations between each log line (listing file)</td>
<td>100</td>
</tr>
<tr>
<td>print level</td>
<td>Amount of information printed during optimization</td>
<td>0</td>
</tr>
<tr>
<td>scale print</td>
<td>Print scaling factors</td>
<td></td>
</tr>
<tr>
<td>solution</td>
<td>Prints MINOS solution</td>
<td>NO</td>
</tr>
<tr>
<td>summary frequency</td>
<td>Number of iterations between each log line (log file)</td>
<td>100</td>
</tr>
</tbody>
</table>

6.2 Tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash tolerance</td>
<td>Allow crash procedure to ignore small elements in eligible columns</td>
<td>0.1</td>
</tr>
<tr>
<td>feasibility tolerance</td>
<td>Feasibility tolerance for linear equations</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>linesearch tolerance</td>
<td>Accuracy required for steplength</td>
<td>0.1</td>
</tr>
<tr>
<td>LU density tolerance</td>
<td>When to use dense factorization</td>
<td>0.5</td>
</tr>
<tr>
<td>LU factor tolerance</td>
<td>Trade-off between stability and sparsity in basis factorization</td>
<td>100.0</td>
</tr>
<tr>
<td>LU singularity tolerance</td>
<td>Protection against ill-conditioned basis matrices</td>
<td>1.0e-11</td>
</tr>
<tr>
<td>LU update tolerance</td>
<td>Trade-off between stability and sparsity in basis factorization</td>
<td>10.0</td>
</tr>
<tr>
<td>optimality tolerance</td>
<td>Reduced gradient optimality check</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>row tolerance</td>
<td>Accuracy requirements for nonlinear rows</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>scale print tolerance</td>
<td>Scale print flag and set tolerance</td>
<td>0.9</td>
</tr>
<tr>
<td>scale tolerance</td>
<td>Scale tolerance</td>
<td>0.9</td>
</tr>
<tr>
<td>subspace tolerance</td>
<td>Determines when nonbasics becomes superbasic</td>
<td>0.5</td>
</tr>
</tbody>
</table>

6.3 Limits

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian dimension</td>
<td>Size of Hessian matrix</td>
<td>1</td>
</tr>
<tr>
<td>iterations limit</td>
<td>Minor iteration limit</td>
<td>1000</td>
</tr>
<tr>
<td>major iterations</td>
<td>Max number of major iterations</td>
<td>50</td>
</tr>
<tr>
<td>minor iterations</td>
<td>Max number of minor iterations between linearizations of nonlinear constraints</td>
<td>40</td>
</tr>
<tr>
<td>superbasics limit</td>
<td>Maximum number of superbasics</td>
<td>1</td>
</tr>
<tr>
<td>unbounded objective value</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e20</td>
</tr>
<tr>
<td>unbounded step size</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e10</td>
</tr>
</tbody>
</table>
6.4 Other algorithmic options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>check frequency</td>
<td>Number of iterations between numerical accuracy check</td>
<td>60</td>
</tr>
<tr>
<td>completion</td>
<td>Completion of subproblems (full/partial)</td>
<td>FULL</td>
</tr>
<tr>
<td>crash option</td>
<td>Controls the basis crash algorithm</td>
<td>3</td>
</tr>
<tr>
<td>expand frequency</td>
<td>Setting for anti-cycling mechanism</td>
<td>10000</td>
</tr>
<tr>
<td>factorization frequency</td>
<td>Number of iterations between basis factorizations</td>
<td>100</td>
</tr>
<tr>
<td>lagrangian</td>
<td>Determines form of objection function in the linearized subproblems</td>
<td>YES</td>
</tr>
<tr>
<td>LU complete pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU partial pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU rook pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>major damping parameter</td>
<td>Prevents large relative changes between subproblem solutions</td>
<td>2.0</td>
</tr>
<tr>
<td>minor damping parameter</td>
<td>Limit change in x during linesearch</td>
<td>2.0</td>
</tr>
<tr>
<td>multiple price</td>
<td>Multiple pricing</td>
<td>1</td>
</tr>
<tr>
<td>partial price</td>
<td>Number of segments in partial pricing strategy</td>
<td>10</td>
</tr>
<tr>
<td>penalty parameter</td>
<td>Used in modified augmented Lagrangian</td>
<td>0.1</td>
</tr>
<tr>
<td>radius of convergence</td>
<td>Determines reduction the penalty parameter</td>
<td>0.01</td>
</tr>
<tr>
<td>scale all variables</td>
<td>Synonym to scale option 2</td>
<td></td>
</tr>
<tr>
<td>scale linear variables</td>
<td>Synonym to scale option 1</td>
<td></td>
</tr>
<tr>
<td>scale no</td>
<td>Synonym to scale option 0</td>
<td></td>
</tr>
<tr>
<td>scale nonlinear variables</td>
<td>Synonym to scale option 2</td>
<td></td>
</tr>
<tr>
<td>scale option</td>
<td>Scaling</td>
<td>1</td>
</tr>
<tr>
<td>scale yes</td>
<td>Synonym to scale option 1</td>
<td></td>
</tr>
<tr>
<td>start assigned nonlinear</td>
<td>Starting strategy when there is no basis</td>
<td>SUPERBASIC</td>
</tr>
<tr>
<td>verify constraint gradients</td>
<td>Synonym to verify level 2</td>
<td></td>
</tr>
<tr>
<td>verify gradients</td>
<td>Synonym to verify level 3</td>
<td></td>
</tr>
<tr>
<td>verify level</td>
<td>Verification of gradients</td>
<td>0</td>
</tr>
<tr>
<td>verify no</td>
<td>Synonym to verify level 0</td>
<td></td>
</tr>
<tr>
<td>verify objective gradients</td>
<td>Synonym to verify level 1</td>
<td></td>
</tr>
<tr>
<td>verify yes</td>
<td>Synonym to verify level 3</td>
<td></td>
</tr>
<tr>
<td>weight on linear objective</td>
<td>Composite objective weight</td>
<td>0.0</td>
</tr>
</tbody>
</table>

6.5 Examples of GAMS/MINOS Option File

The following example illustrates the use of certain options that might be helpful for "difficult" models involving nonlinear constraints. Experimentation may be necessary with the values specified, particularly if the sequence of major iterations does not converge using default values.

* These options might be relevant for very nonlinear models.

Major damping parameter 0.2 * may prevent divergence.
Minor damping parameter 0.2 * if there are singularities
* in the nonlinear functions.
Penalty parameter 10.0 * or 100.0 perhaps a value
* higher than the default.
Scale linear variables * (This is the default.)
Conversely, nonlinearly constrained models that are very nearly linear may optimize more efficiently if some of the cautious defaults are relaxed:

* Suggestions for models with MILDLY nonlinear constraints
  Completion Full
  Penalty parameter 0.0 * or 0.1 perhaps-a value
  * smaller than the default.
  * Scale one of the following
  Scale all variables * if starting point is VERY GOOD.
  Scale linear variables * if they need it.
  Scale No * otherwise.

Most of the options described in the next section should be left at their default values for any given model. If experimentation is necessary, we recommend changing just one option at a time.

7 Special Notes

7.1 Modeling Hints

Unfortunately, there is no guarantee that the algorithm just described will converge from an arbitrary starting point. The concerned modeler can influence the likelihood of convergence as follows:

- Specify initial activity levels for the nonlinear variables as carefully as possible (using the GAMS suffix .L).
- Include sensible upper and lower bounds on all variables.
- Specify a Major damping parameter that is lower than the default value, if the problem is suspected of being highly nonlinear.
- Specify a Penalty parameter ρ that is higher than the default value, again if the problem is highly nonlinear.

In rare cases it may be safe to request the values λ_k = 0 and ρ = 0 for all subproblems, by specifying Lagrangian=No. However, convergence is much more like with the default setting, Lagrangian=Yes.

The initial estimate of the Lagrange multipliers is then λ_0 = 0, but for later subproblems λ_k is taken to be the Lagrange multipliers associated with the (linearized) nonlinear constraints at the end of the previous major iteration.

For the first subproblem, the default value for the penalty parameter is ρ = 100.0/m_1 where m_1 is the number of nonlinear constraints. For later subproblems, ρ is reduced in stages when it appears that the sequence \{x_k, λ_k\} is converging. In many times it is safe to specify λ = 0, particularly if the problem is only mildly nonlinear. This may improve the overall efficiency.

7.2 Storage

GAMS/MINOS uses one large array of main storage for most of its workspace. The implementation places no fixed limit on the size of a problem or on its shape (many constraints and relatively few variables, or vice versa). In general, the limiting factor will be the amount of main storage available on a particular machine, and the amount of computation time that one’s budget and/or patience can stand.

Some detailed knowledge of a particular model will usually indicate whether the solution procedure is likely to be efficient. An important quantity is m, the total number of general constraints in (2) and (3). The amount of workspace required by GAMS/MINOS is roughly 100m words, where one "word" is the relevant storage unit for the floating-point arithmetic being used. This usually means about 800m bytes for workspace. A further 300K bytes, approximately, are needed for the program itself, along with buffer space for several files. Very roughly, then, a model with m general constraints requires about \(m + 300\) K bytes of memory.

Another important quantity, is n, the total number of variables in x and y. The above comments assume that n is not much larger than $n/m$, the number of constraints. A typical ratio for n/m is 2 or 3.
If there are many nonlinear variables (i.e., if \( n_1 \) is large), much depends on whether the objective function or the constraints are highly nonlinear or not. The degree of nonlinearity affects \( s \), the number of superbasic variables. Recall that \( s \) is zero for purely linear problems. We know that \( s \) need never be larger than \( n_1 + 1 \). In practice, \( s \) is often very much less than this upper limit.

In the quasi-Newton algorithm, the dense triangular matrix \( R \) has dimension \( ss \) and requires about \( s^2/2 \) words of storage. If it seems likely that \( s \) will be very large, some aggregation or reformulation of the problem should be considered.

## 8 The GAMS/MINOS Log File

MINOS writes different logs for LPs, NLPs with linear constraints, and NLPs with non-linear constraints. In this section, a sample log file is shown for each case, and the appearing messages are explained.

More information can be found at [Anatomy of the MINOS Log File](#).

### 8.1 Linear Programs

MINOS uses a standard two-phase Simplex method for LPs. In the first phase, the sum of the infeasibilities at each iteration is minimized. Once feasibility is attained, MINOS switches to phase 2 where it minimizes (or maximizes) the original objective function. The different objective functions are called the phase 1 and phase 2 objectives. Notice that the marginals in phase 1 are with respect to the phase 1 objective. This means that if MINOS interrupts in phase 1, the marginals are "wrong" in the sense that they do not reflect the original objective.

The log for the problem **TURKPOW** is as follows:

```
GAMS Rev 235 Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- turkpow.gms(230) 3 Mb
--- Starting execution: elapsed 0:00:00.009
--- turkpow.gms(202) 4 Mb
--- Generating LP model turkey
--- turkpow.gms(205) 4 Mb
--- 350 rows 949 columns 5,872 non-zeroes
--- Executing MINOS: elapsed 0:00:00.025

GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S 5.51 (Jun 2004)

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated -- 1.60 Mb

Reading Rows...
Reading Columns...

<table>
<thead>
<tr>
<th>Itn</th>
<th>ninf</th>
<th>sinf</th>
<th>objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3</td>
<td>2.283E-01</td>
<td>-2.51821463E+04</td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0.000E+00</td>
<td>2.02819284E+04</td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>0.000E+00</td>
<td>1.54107277E+04</td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>0.000E+00</td>
<td>1.40211808E+04</td>
</tr>
<tr>
<td>500</td>
<td>0</td>
<td>0.000E+00</td>
<td>1.33804183E+04</td>
</tr>
</tbody>
</table>
```
The first line that is written by MINOS is the version string: GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows This line identifies which version of the MINOS libraries and links you are using, and is only to be deciphered by GAMS support personnel.

After some advertisement text we see the amount of work space that is allocated: 2.08 Mb When MINOS is loaded, the amount of memory needed is first estimated. This estimate is based on statistics like the number of rows, columns and non-zeros. This amount of memory is then allocated and the problem is then loaded into MINOS.

The columns have the following meaning:

Itn
   Iteration number.

ninf
   Number of infeasibilities. If nonzero the model is still infeasible.

sinf
   The sum of the infeasibilities. This number is minimized during Phase I. Once the model is feasible this number is zero.

objective
   The value of the objective function: \( z = \sum c_i x_i \). In phase II this number is maximized or minimized. In phase I it may move in the wrong direction.

The final line indicates the exit status of MINOS.

### 8.2 Linearly Constrained NLP’s

The log is basically the same as for linear models. The only difference is that not only matrix row and columns need to be loaded, but also instructions for evaluating functions and gradients.

The log for the problem WEAPONS is as follows:

GAMS Rev 235 Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- weapons.gms(77) 3 Mb
--- Starting execution: elapsed 0:00:00.005
--- weapons.gms(66) 4 Mb
--- Generating NLP model war
--- weapons.gms(68) 6 Mb
--- 13 rows 66 columns 156 non-zeros
--- 706 nl-code 65 nl-non-zeros
--- weapons.gms(68) 4 Mb
--- Executing MINOS: elapsed 0:00:00.013

GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
MINOS 5.51 (Jun 2004)
8.3 NLP’s with Nonlinear Constraints

For models with nonlinear constraints the log is more complicated. **CAMCGE** from the model library is such an example, and the screen output resulting from running it is shown below:

```
8.3 NLP's with Nonlinear Constraints

For models with nonlinear constraints the log is more complicated. CAMCGE from the model library is such an example, and the screen output resulting from running it is shown below:

GAMS Rev 235 Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- camcge.gms(450) 3 Mb
--- Starting execution: elapsed 0:00:00.010
--- camcge.gms(441) 4 Mb
--- Generating NLP model camcge
--- camcge.gms(450) 6 Mb
--- 243 rows 280 columns 1,356 non-zeroes
--- 5,524 nl-code 850 nl-non-zeroes
--- camcge.gms(450) 4 Mb
--- Executing MINOS: elapsed 0:00:00.023
```
Two sets of iterations - Major and Minor, are now reported. A description of the various columns present in this log file follows:

**Major**
A major iteration involves linearizing the nonlinear constraints and performing a number of minor iterations on the resulting subproblem. The objective for the subproblem is an augmented Lagrangian, not the true objective function.

**minor**
The number of minor iterations performed on the linearized subproblem. If it is a simple number like 90, then the subproblem was solved to optimality. Here, $2T$ means that the subproblem was terminated. In general the $T$ is not something to worry about. Other possible flags are $I$ and $U$, which mean that the subproblem was Infeasible or Unbounded. MINOS may have difficulty if these keep occurring.

**step**
The step size taken towards the solution suggested by the last major iteration. Ideally this should be 1.0, especially near an optimum. If the subproblem solutions are widely different, MINOS may reduce the step size under control of the *Major Damping parameter*.

**objective**
The objective function for the original nonlinear program.

**Feasible**
Primal infeasibility, indicating the maximum non-linear constraint violation.

**Optimal**
The maximum dual infeasibility, measured as the maximum departure from complementarity. If we call $d_j$ the reduced cost of variable $x_j$, then the dual infeasibility of $x_j$ is $d_j \times \min\{x_j - \ell_j, 1\}$ or $-d_j \times \min\{u_j - x_j, 1\}$ depending on the sign of $d_j$.

**nsb**
Number of superbasics. If the model is feasible this number cannot exceed the superbasic limit, which may need to be reset to a larger number if the numbers in this column become larger.

**ncon**
The number of times MINOS has evaluated the nonlinear constraints and their derivatives.

**penalty**
The current value of the penalty parameter in the augmented Lagrangian (the objective for the subproblems). If the major iterations appear to be converging, MINOS will decrease the penalty parameter. If there appears to be difficulty, such as unbounded subproblems, the penalty parameter will be increased.

**BSswp**
Number of basis swaps: the number of $(B \leftrightarrow S)$ (i.e. basic vs. superbasic) changes.

---

EXIT - Optimal Solution found, objective: 191.7346

--- Restarting execution
--- camcge.gms(450) 0 Mb
--- Reading solution for model camcge
*** Status: Normal completion
Note: The CAMCGE model (like many CGE models or other almost square systems) can better be solved with the MINOS option Start Assigned Nonlinears Basic.

9 Detailed Description of MINOS Options

The following is an alphabetical list of the keywords that may appear in the GAMS/MINOS options file, and a description of their effect. The letters $i$ and $r$ denote integer and real values. The number $\delta$ denotes machine precision (typically $10^{-15}$ or $10^{-16}$). Options not specified will take the default values shown.

**check frequency (integer):** Number of iterations between numerical accuracy check

Every $i$th iteration after the most recent basis factorization, a numerical test is made to see if the current solution $x$ satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax + s = 0$ where $s$ is the set of slack variables. To perform the numerical test, the residual vector $r = Ax + s$ is computed. If the largest component of $r$ is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

(default = 60)

**completion (string):** Completion of subproblems (full/partial)

When there are nonlinear constraints, this determines whether subproblems should be solved to moderate accuracy (partial completion), or to full accuracy (full completion), GAMS/MINOS implements the option by using two sets of convergence tolerances for the subproblems. Use of partial completion may reduce the work during early major iterations, unless the Minor iterations limit is active. The optimal set of basic and superbasic variables will probably be determined for any given subproblem, but the reduced gradient may be larger than it would have been with full completion. An automatic switch to full completion occurs when it appears that the sequence of major iterations is converging. The switch is made when the nonlinear constraint error is reduced below $100 \times$ (Row tolerance), the relative change in $\Lambda_k$ is 0.1 or less, and the previous subproblem was solved to optimality. Full completion tends to give better Langrange-multiplier estimates. It may lead to fewer major iterations, but may result in more minor iterations.

(default = FULL)

- FULL Solve subproblems to full accuracy
- PARTIAL Solve subproblems to moderate accuracy

**crash option (integer):** Controls the basis crash algorithm

If a restart is not being performed, an initial basis will be selected from certain columns of the constraint matrix ($A I$). The value of the parameter $i$ determines which columns of $A$ are eligible. Columns of $I$ are used to fill gaps where necessary. If $i > 0$, three passes are made through the relevant columns of $A$, searching for a basis matrix that is essentially triangular. A column is assigned to pivot on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis). Pass 1 selects pivots from free columns (corresponding to variables with no upper and lower bounds). Pass 2 requires pivots to be in rows associated with equality (=E=) constraints. Pass 3 allows the pivots to be in inequality rows. For remaining (unassigned) rows, the associated slack variables are inserted to complete the basis.

(default = 3)

- 0 Initial basis will be a slack basis
- 1 All columns are eligible
- 2 Only linear columns are eligible
- 3 Columns appearing nonlinearly in the objective are not eligible
- 4 Columns appearing nonlinearly in the constraints are not eligible

**crash tolerance (real):** Allow crash procedure to ignore small elements in eligible columns

The Crash tolerance $r$ allows the starting procedure CRASH to ignore certain small nonzeros in each column of
If $a_{\text{max}}$ is the largest element in column $j$, other nonzeros $a_{ij}$ in the column are ignored if $|a_{ij}| < a_{\text{max}} \times r$. To be meaningful, the parameter $r$ should be in the range $0 \leq r < 1$. When $r > 0.0$ the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of $A$ and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems. For example, suppose the first $m$ columns of $A$ are the matrix shown under LU tolerance; i.e., a tridiagonal matrix with entries -1, 4, -1. To help CRASH choose all $m$ columns for the initial basis, we could specify Crash tolerance $r$ for some value of $r > 0.25$.

(debug level (integer): Controls amount of debug information written)

This causes various amounts of information to be output. Most debug levels will not be helpful to GAMS users, but they are listed here for completeness. Note that you will need to use the GAMS statement `OPTION SYSOUT=on;` to echo the MINOS listing to the GAMS listing file.

- **debug level 0**: No debug output.
- **debug level 2**(or more): Output from `M5SETX` showing the maximum residual after a row check.
- **debug level 40**: Output from `LU8RPC` (which updates the LU factors of the basis matrix), showing the position of the last nonzero in the transformed incoming column.
- **debug level 50**: Output from `LU1MAR` (which updates the LU factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.
- **debug level 100**: Output from `M2BFAC` and `M5LOG` listing the basic and superbasic variables and their values at every iteration.

(expand frequency (integer): Setting for anti-cycling mechanism)

This option is part of anti-cycling procedure designed to guarantee progress even on highly degenerate problems. For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose the specified feasibility tolerance is $\delta$. Over a period of $i$ iterations, the tolerance actually used by GAMS/MINOS increases from $0.5\times\delta$ to $\delta$ (in steps $0.5\times\delta/i$). For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced. Increasing $i$ helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see Pivot tolerance).

(factorization frequency (integer): Number of iterations between basis factorizations)

At most $i$ basis changes will occur between factorizations of the basis matrix. With linear programs, the basis factors are usually updated every iteration. The default $i$ is reasonable for typical problems. Higher values up to $i = 100$ (say) may be more efficient on problems that are extremely sparse and well scaled. When the objective function is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the Check frequency) to ensure that the general constraints are satisfied. If necessary the basis will be re-factorized before the limit of $i$ updates is reached. When the constraints are nonlinear, the Minor iterations limit will probably preempt $i$.

(feasibility tolerance (real): Feasibility tolerance for linear equations)
When the constraints are linear, a feasible solution is one in which all variables, including slacks, satisfy their upper and lower bounds to within the absolute tolerance \( r \). (Since slacks are included, this means that the general linear constraints are also satisfied within \( r \).) GAMS/MINOS attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible. Let \( SINF \) be the corresponding sum of infeasibilities. If \( SINF \) is quite small, it may be appropriate to raise \( r \) by a factor of 10 or 100. Otherwise, some error in the data should be suspected. If \( SINF \) is not small, there may be other points that have a significantly smaller sum of infeasibilities. GAMS/MINOS does not attempt to find a solution that minimizes the sum. If \( Scale \) option = 1 or 2, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful). A nonlinear objective function \( F(x) \) will be evaluated only at feasible points. If there are regions where \( F(x) \) is undefined, every attempt should be made to eliminate these regions from the problem. For example, for a function \( F(x) = \sqrt{x_1} + \log(x_2) \), it should be essential to place lower bounds on both variables. If \( Feasibility \) tolerance \( = 10^{-6} \), the bounds \( x_1 > 10^{-5} \) and \( x_2 > 10^{-4} \) might be appropriate. (The log singularity is more serious; in general, keep variables as far away from singularities as possible.) If the constraints are nonlinear, the above comments apply to each major iteration. A feasible solution satisfies the current linearization of the constraints to within the tolerance \( r \). The associated subproblem is said to be feasible. As for the objective function, bounds should be used to keep \( x \) more than \( r \) away from singularities in the constraint functions \( f(x) \). At the start of major iteration \( k \), the constraint functions \( f(x_k) \) are evaluated at a certain point \( x_k \). This point always satisfies the relevant bounds \((l < x_k < u)\), but may not satisfy the general linear constraints. During the associated minor iterations, \( F(x_k) \) and \( f(x_k) \) will be evaluated only at points \( x \) that satisfy the bound and the general linear constraints (as well as the linearized nonlinear constraints).

If a subproblem is infeasible, the bounds on the linearized constraints are relaxed temporarily, in several stages. Feasibility with respect to the nonlinear constraints themselves is measured against the Row tolerance (not against \( r \)). The relevant test is made at the start of a major iteration.

(default = \( 1.0 \times 10^{-6} \))

**hessian dimension (integer):** Size of Hessian matrix

This specifies than an \( r \times r \) triangular matrix \( R \) is to be available for use by the quasi-Newton algorithm (to approximate the reduced Hessian matrix according to \( Z^T H Z \approx R^T R \)). Suppose there are \( s \) superbasic variables at a particular iteration. Whenever possible, \( r \) should be greater than \( s \). If \( r > s \), the first \( s \) columns of \( R \) will be used to approximate the reduced Hessian in the normal manner. If there are no further changes to the set of superbasic variables, the rate of convergence will ultimately be superlinear. If \( r < s \), a matrix of the form,

\[
R = \text{diag}(R_r, D)
\]

will be used to approximate the reduced Hessian, where \( R_r \) is an \( r \times r \) upper triangular matrix and \( D \) is a diagonal matrix of order \( s - r \). The rate of convergence will no longer be superlinear (and may be arbitrarily slow). The storage required if of the order \( \sqrt{s} r / 2 \), which is substantial if \( r \) is as large as 200 (say). In general, \( r \) should be slightly over-estimate of the final number of superbasic variables, whenever storage permits. It need not be larger than \( n_1 + 1 \), where \( n_1 \) is the number of nonlinear variables. For many problems it can be much smaller than \( n_1 \). If **Superbasics limit** \( s \) is specified, the default value of \( r \) is the same number, \( s \) (and conversely). This is a safeguard to ensure super-linear convergence wherever possible. If neither \( r \) nor \( s \) is specified, GAMS chooses values for both, using certain characteristics of the problem.

(default = 1)

**iterations limit (integer):** Minor iteration limit

This is maximum number of minor iterations allowed (i.e., iterations of the simplex method or the reduced-gradient method). This option, if set, overrides the GAMS **ITERLIM** specification. If \( i = 0 \), no minor iterations are performed, but the starting point is tested for both feasibility and optimality. Iters or Itns are alternative keywords.

(default = 1000)

**lagrangian (string):** Determines form of objection function in the linearized subproblems

This determines the form of the objective function used for the linearized subproblems. The default value **yes** is highly recommended. The **Penalty parameter** value is then also relevant. If **No** is specified, the nonlinear constraint functions will be evaluated only twice per major iteration. Hence this option may be useful if the nonlinear constraints are very expensive to evaluate. However, in general there is a great risk that convergence
may not occur.

(default = YES)

NO Nondefault value (not recommended)
YES Default value (recommended)

**linesearch tolerance** *(real)*: Accuracy required for steplength

For nonlinear problems, this controls the accuracy with which a steplength \(\alpha\) is located in the one-dimensional problem

\[
\begin{align*}
\text{minimize } & F(x + \alpha p) \\
\text{subject to } & 0 < \alpha \leq \beta 
\end{align*}
\]

A linesearch occurs on most minor iterations for which \(x\) is feasible. (If the constraints are nonlinear, the function being minimized is the augmented Lagrangian.) \(r\) must be a real value in the range \(0.0 < r < 1.0\). The default value \(r = 0.1\) requests a moderately accurate search. It should be satisfactory in most cases. If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate: try \(r = 0.01\) or \(r = 0.001\). The number of iterations should decrease, and this will reduce total run time if there are many linear or nonlinear constraints. If the nonlinear function are expensive to evaluate, a less accurate search may be appropriate; try \(r = 0.5\) or perhaps \(r = 0.9\). (The number of iterations will probably increase but the total number of function evaluations may decrease enough to compensate.)

(default = 0.1)

**log frequency** *(integer)*: Number of iterations between each log line (listing file)

In general, one line of the iteration log is printed every \(i\)th minor iteration. A heading labels the printed items, which include the current iteration number, the number and sum of feasibilities (if any), the subproblem objective value (if feasible), and the number of evaluations of the nonlinear functions. A value such as \(i = 10, 100\) or larger is suggested for those interested only in the final solution. *Log frequency 0* may be used as shorthand for *Log frequency 99999*. If *Print level > 0*, the default value of \(i\) is 1. If *Print level = 0*, the default value of \(i\) is 100. If *Print level = 0* and the constraints are nonlinear, the minor iteration log is not printed (and the *Log frequency* is ignored). Instead, one line is printed at the beginning of each major iteration.

(default = 100)

**LU complete pivoting** *(string)*: LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook and complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say \(< 2.0\)).

**LU density tolerance** *(real)*: When to use dense factorization

The density tolerance is used during LUSOL’s basis factorization \(B=LU\). Columns of \(L\) and rows of \(U\) are formed one at the time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds this tolerance, the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense \(LU\) procedure. Raising the tolerance towards 1.0 may give slightly sparser factors, with a slight increase in factorization time.

(default = 0.5)

**LU factor tolerance** *(real)*: Trade-off between stability and sparsity in basis factorization

This tolerances affect the stability and sparsity of the basis factorization \(B = LU\) during factorization. The value \(r\) specified must satisfy \(r \geq 1.0\).

- The default value \(r = 100.0\) usually strikes a good compromise between stability and sparsity.
- For large and relatively dense problems, a larger value may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set \(r\) to a value smaller than the default in order to achieve stability.
**LU partial pivoting** *(string)*: LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say < 2.0).

**LU rook pivoting** *(string)*: LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say < 2.0).

**LU singularity tolerance** *(real)*: Protection against ill-conditioned basis matrices

When the basis is refactorized, the diagonal elements of $U$ are tested as follows: if $|U_{jj}| < r$ or $|U_{jj}| < r \times \max_i |U_{ij}|$, the $j$th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.) In some cases, the Jacobian matrix may converge to values that make the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting $r = 1.0 \times 10^{-5}$, say, may help cause a judicious change of basis. (default = 1.0e-11)

**LU update tolerance** *(real)*: Trade-off between stability and sparsity in basis factorization

This tolerances affect the stability and sparsity of the basis factorization $B = LU$ during updates. The value $r$ specified must satisfy $r \geq 1.0$.

- The default value $r = 10.0$ usually strikes a good compromise between stability and sparsity.
- For large and relatively dense problems, $r = 25.0$ (say) may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set $r$ to a value smaller than the default in order to achieve stability.

(designated = 10.0)

**major damping parameter** *(real)*: Prevents large relative changes between subproblem solutions

The parameter may assist convergence on problems that have highly nonlinear constraints. It is intended to prevent large relative changes between subproblem solutions $(x_k, \lambda_k)$ and $(x_{k+1}, \lambda_{k+1})$. For example, the default value 2.0 prevents the relative change in either $x_k$ or $\lambda_k$ from exceeding 200 percent. It will not be active on well-behaved problems. The parameter is used to interpolate between the solutions at the beginning and end of each major iteration. Thus $x_{k+1}$ and $\lambda_{k+1}$ are changed to $x_k + \text{sigma} \times (x_{k+1} - x_k)$ and $\lambda_k + \text{sigma} \times (\lambda_{k+1} - \lambda_k)$ for some step-length $\text{sigma} < 1$. In the case of nonlinear equation (where the number of constraints is the same as the number of variables) this gives a damped Newton method. This is very crude control. If the sequence of major iterations does not appear to be converging, one should first re-run the problem with a higher Penalty parameter (say 10 or 100 times the default $\rho$). (Skip this re-run in the case of nonlinear equations: there are no degrees of freedom and the value of $\rho$ is irrelevant.) If the subproblem solutions continue to change violently, try reducing $r$ to 0.2 or 0.1 (say). For implementation reason, the shortened step to $\text{sigma}$ applies to the nonlinear variables $x$, but not to the linear variables $y$ or the slack variables $s$. This may reduce the efficiency of the control.

(designated = 2.0)

**major iterations** *(integer)*: Max number of major iterations

This is maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations may not converge. The progress of the major iterations can be best monitored using *Print level 0* (the default).

(designated = 50)

**minor damping parameter** *(real)*: Limit change in $x$ during linesearch
This parameter limits the change in $x$ during a linesearch. It applies to all nonlinear problems, once a feasible solution or feasible subproblem has been found. A linesearch of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

is performed over the range $0 < \alpha \leq \beta$, where $\beta$ is the step to the nearest upper or lower bound on $x$. Normally, the first step length tried is $\alpha_1 = \min(1, \beta)$. In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax + b$, even a moderate change in the components of $r$ can lead to floating-point overflow. The parameter $r$ is therefore used to define a limit

$$\beta'' = r(1 + ||x||p||)$$

and the first evaluation of $F(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \beta, \beta'')$. Wherever possible, upper and lower bounds on $x$ should be used to prevent evaluation of nonlinear functions at meaningless points. The *Minor damping parameter* provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or $0.01$ may be helpful when rapidly varying function are present. A good starting point may be required. An important application is to the class of nonlinear least squares problems. In case where several local optima exist, specifying a small value for $r$ may help locate an optima near the starting point.

(default = 2.0)

**minor iterations (integer):** Max number of minor iterations between linearizations of nonlinear constraints

This is the maximum number of feasible minor iterations allowed between successive linearizations of the nonlinear constraints. A moderate value (e.g., $20 \leq i \leq 50$) prevents excessive efforts being expended on early major iterations, but allows later subproblems to be solved to completion. The limit applies to both infeasible and feasible iterations. In some cases, a large number of iterations, (say $K$) might be required to obtain a feasible subproblem. If good starting values are supplied for variables appearing nonlinearly in the constraints, it may be sensible to specify $> K$, to allow the first major iteration to terminate at a feasible (and perhaps optimal) subproblem solution. (If a good initial subproblem is arbitrarily interrupted by a small $i$th subsequent linearization may be less favorable than the first.) In general it is unsafe to specify value as small as $i = 1$ or 2 even when an optimal solution has been reached, a few minor iterations may be needed for the corresponding subproblem to be recognized as optimal. The *Iteration limit* provides an independent limit on the total minor iterations (across all subproblems). If the constraints are linear, only the *Iteration limit* applies: the minor iterations value is ignored.

(default = 40)

**multiple price (integer):** Multiple pricing

Pricing refers to a scan of the current non-basic variables to determine if any should be changed from their value (by allowing them to become superbasic or basic). If multiple pricing in effect, the $i$ best non-basic variables are selected for admission of appropriate sign. If partial pricing is also in effect, the best $i$ best variables are selected from the current partition of $A$ and $I$. The default $i = 1$ is best for linear programs, since an optimal solution will have zero superbasic variables. Warning: If $i > 1$, GAMS/MINOS will use the reduced-gradient method (rather than the simplex method) even on purely linear problems. The subsequent iterations do not correspond to the efficient minor iterations carried out by commercial linear programming system using multiple pricing. (In the latter systems, the classical simplex method is applied to a tableau involving $i$ dense columns of dimension $m$, and $i$ is therefore limited for storage reasons typically to the range $2 \leq i \leq 7$.) GAMS/MINOS varies all superbasic variables simultaneously. For linear problems its storage requirements are essentially independent of $i$. Larger values of $i$ are therefore practical, but in general the iterations and time required when $i > 1$ are greater than when the simplex method is used ($i = 1$). On large nonlinear problems it may be important to set $i > 1$ if the starting point does not contain many superbasic variables. For example, if a problem has 3000 variables and 500 of them are nonlinear, the optimal solution may well have 200 variables superbasic. If the problem is solved in several runs, it may be beneficial to use $i = 10$ (say) for early runs, until it seems that the number of superbasics has leveled off. If *Multiple price* $i$ is specified, it is also necessary to specify *Superbasic limit* $s$ for some $s > i$.

(default = 1)

**optimality tolerance (real):** Reduced gradient optimality check

This is used to judge the size of the reduced gradients $d_j = g_j - pi^T a_j$, where $g_j$ is the gradient of the objective
function corresponding to the \( p^\text{th} \) variable. \( a_j \) is the associated column of the constraint matrix (or Jacobian), and \( pi \) is the set of dual variables. By construction, the reduced gradients for basic variables are always zero. Optimality will be declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy 
\[ d_j / ||pi|| \geq -r \text{ or } d_j / ||pi|| \leq r \] respectively, and if \( d_j / ||pi|| \leq r \) for superbasic variables. In the \( ||pi|| \) is a measure of the size of the dual variables. It is included to make the tests independents of a scale factor on the objective function. The quantity actually used is defined by
\[ sigma = \text{sum}(i, \text{abs}(pi(i))), \quad ||pi|| = \max\{sigma/sqrt(m),1\} \]
so that only large scale factors are allowed for. If the objective is scaled down to be small, the optimality test effectively reduced to comparing \( D_j \) against the tolerance \( r \).

(default = 1.0e-6)

**partial price (integer):** Number of segments in partial pricing strategy

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each pricing operation (when a nonbasic variable is selected to become basic or superbasic). When \( i = 1 \), all columns of the constraints matrix \((A I)\) are searched. Otherwise, \( A_i \) and \( I \) are partitioned to give \( i \) roughly equal segments \( A_i, I_j \) \((j = 1 \text{ to } i)\). If the previous search was successful on \( A_{j-1}, I_{j-1} \), the next search begins on the segments \( A_i, I_j \) (All subscripts here are modulo \( i \)). If a reduced gradient is found that is large than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. (Several may be selected if multiple pricing has been specified.) If nothing is found, the search continues on the next segments \( A_{j+1}, I_{j+1} \) and so on. Partial price \( t \) (or \( t/2 \) or \( t/3 \)) may be appropriate for time-stage models having \( t \) time periods

(default = 10)

**penalty parameter (real):** Used in modified augmented Lagrangian

This specifies the value of \( rho \) in the modified augmented Lagrangian. It is used only when \( \text{Lagrangian} = \text{yes} \) (the default setting). For early runs on a problem is known to be unknown characteristics, the default value should be acceptable. If the problem is problem is known to be highly nonlinear, specify a large value, such as 10 times the default. In general, a positive value of \( rho \) may be necessary of known to ensure convergence, even for convex programs. On the other hand, if \( rho \) is too large, the rate of convergence may be unnecessarily slow. If the functions are not highly nonlinear or a good starting point is known, it will often be safe to specify penalty parameter 0.0. Initially, use a moderate value for \( r \) (such as the default) and a reasonably low \( \text{Iterations} \) and/or \( \text{major iterations limit} \). If successive major iterations appear to be terminating with radically different solutions, the penalty parameter should be increased. (See also the \( \text{Major damping parameter} \)). If there appears to be little progress between major iteration, it may help to reduce the penalty parameter.

(default = 0.1)

**print level (integer):** Amount of information printed during optimization

This varies the amount of information that will be output during optimization. Print level 0 sets the default Log and summary frequencies to 100. It is then easy to monitor the progress of run. Print level 1 (or more) sets the default Log and summary frequencies to 1, giving a line of output for every minor iteration. Print level 1 also produces basis statistics, i.e., information relating to LU factors of the basis matrix whenever the basis is re-factorized. For problems with nonlinear constraints, certain quantities are printed at the start of each major iteration. The value of is best thought of as a binary number of the form

\[ \text{Print level JFLXB} \]

where each letter stand for a digit that is either 0 or 1. The quantities referred to are:

- **B** Basis statistics, as mentioned above
- **X** \( x_k \), the nonlinear variables involved in the objective function or the constraints.
- **L** \( \lambda_k \), the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if \( \text{Lagrangian}=\text{No} \), since then \( \lambda_k = 0 \).)
- **F** \( f(x_k) \), the values of the nonlinear constraint functions.
- **J** \( J(x_k) \), the Jacobian matrix.
To obtain output of any item, set the corresponding digit to 1, otherwise to 0. For example, *Print level 10* sets \( X = 1 \) and the other digits equal to zero; the nonlinear variables will be printed each major iteration. If \( J = 1 \), the Jacobian matrix will be output column-wise at the start of each major iteration. Column \( j \) will be preceded by the value of the corresponding variable \( x_j \) and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if \( J = 1 \), there is no reason to specify \( X = 1 \) unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

\[ 3 \ 1.250000D+01 \ \text{BS} \ 1 \ 1.00000D+00 \ 4 \ 2.00000D+00 \]

which would mean that \( x_3 \) is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4. (Note: the GAMS/MINOS row numbers are usually different from the GAMS row numbers; see the Solution option.)

(default = 0)

**radius of convergence (real)**: Determines reduction the penalty parameter

This determines when the penalty parameter \( \rho \) will be reduced (if initialized to a positive value). Both the nonlinear constraint violation (see \( \text{ROWERR} \) below) and the relative change in consecutive Lagrange multiplier estimate must be less than \( r \) at the start of a major iteration before \( \rho \) is reduced or set to zero. A few major iterations later, full completion will be requested if not already set, and the remaining sequence of major iterations should converge quadratically to an optimum.

(default = 0.01)

**row tolerance (real)**: Accuracy requirements for nonlinear rows

This specifies how accurately the nonlinear constraints should be satisfied at a solution. The default value is usually small enough, since model data is often specified to about that an accuracy. Let \( \text{ROWERR} \) be the maximum component of the residual vector \( f(x) + A_j y - b_j \), normalized by the size of the solution. Thus

\[
\text{ROWERR} = \frac{||f(x) + A_j y - b_j||}{1 + X\text{NORM}}
\]

where \( X\text{NORM} \) is a measure of the size of the current solution \((x, y)\). The solution is regarded acceptably feasible if \( \text{ROWERR} \leq r \). If the problem functions involve data that is known to be of low accuracy, a larger Row tolerance may be appropriate.

(default = 1.0e-6)

**scale all variables (string)**: Synonym to scale option 2

**scale linear variables (string)**: Synonym to scale option 1

**scale no (string)**: Synonym to scale option 0

**scale nonlinear variables (string)**: Synonym to scale option 2

**scale option (integer)**: Scaling

*Scale Yes* sets the default. (Caution: If all variables are nonlinear, *Scale Yes* unexpectedly does nothing, because there are no linear variables to scale). *Scale No* suppresses scaling (equivalent to *Scale Option 0*). If nonlinear constraints are present, *Scale option 1* or 0 should generally be rid at first. *Scale option 2* gives scales that depend on the initial Jacobian, and should therefore be used only if (a) good starting point is provided, and (b) the problem is not highly nonlinear.

(default = 1)

0 No scaling If storage is at a premium, this option should be used.

1 Scale linear variables Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see [5]). This will sometimes improve the performance of the solution procedures. *Scale linear variables* is an equivalent option.

2 Scale linear + nonlinear variables All constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side \( b \) or the solution \( x \) is large. This takes into account columns of \((A I)\) that are fixed or have positive lower bounds or negative upper bounds. *Scale nonlinear variables* or *Scale all variables* are equivalent options.
scale print (string): Print scaling factors

This causes the row-scales \( r(i) \) and column-scales \( c(j) \) to be printed. The scaled matrix coefficients are \( a_{ij}^{\prime} = a_{ij} r(i) c(j) \), and the scaled bounds on the variables, and slacks are \( l_{ij}^{\prime} = l_{ij} c(j) \), \( u_{ij}^{\prime} = u_{ij} c(j) \), where \( c(j) = r(j - n) \) if \( j > n \). If a Scale option has not already been specified, Scale print sets the default scaling.

scale print tolerance (real): Scale print flag and set tolerance

See Scale Tolerance. This option also turns on printing of the scale factors.

(default = 0.9)

scale tolerance (real): Scale tolerance

All forms except Scale option may specify a tolerance \( r \) where \( 0 < r < 1 \) (for example: Scale Print Tolerance = 0.99). This affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ration of the largest and smallest nonzero coefficients in each column:

\[
\rho_j = \max_i \frac{|a_{ij}|}{\min_i |a_{ij}|} \quad (a_{ij} \neq 0)
\]

If \( \rho_j \) is less than \( r \) times its previous value, another scaling pass is performed to adjust the row and column scales. Raising \( r \) from 0.9 to 0.99 (say) usually increases the number of scaling passes through \( A \). At most 10 passes are made. If a Scale option has not already been specified, Scale tolerance sets the default scaling.

(default = 0.9)

scale yes (string): Synonym to scale option 1

solution (string): Prints MINOS solution

This controls whether or not GAMS/MINOS prints the final solution obtained. There is one line of output for each constraint and variable. The lines are in the same order as in the GAMS solution, but the constraints and variables labeled with internal GAMS/MINOS numbers rather than GAMS names. (The numbers at the left of each line are GAMS/MINOS column numbers, and those at the right of each line in the rows section are GAMS/MINOS slacks.) The GAMS/MINOS solution may be useful occasionally to interpret certain messages that occur during the optimization, and to determine the final status of certain variables (basic, superbasic or nonbasic).

(default = NO)

NO Turn off printing of solution

YES Turn on printing of solution

start assigned nonlinear (string): Starting strategy when there is no basis

This option affects the starting strategy when there is no basis (i.e., for the first solve or when the GAMS statement option bratio = 1 is used to reject an existing basis.) This option applies to all nonlinear variables that have been assigned nondefault initial values and are strictly between their bounds. Free variables at their default value of zero are excluded. Let \( K \) denote the number of such assigned nonlinear variables.

(default = SUPERBASIC)

SUPERBASIC Default Specify superbasic for highly nonlinear models, as long as \( K \) is not too large (say \( K < 100 \)) and the initial values are good.

BASIC Good for square systems Specify basic for models that are essentially square (i.e., if there are about as many general constraints as variables).

NONBASIC Specify nonbasic if \( K \) is large.

ELIGIBLE FOR CRASH Specify Eligible for Crash for linear or nearly linear models. The nonlinear variables will be treated in the manner described under Crash option.

subspace tolerance (real): Determines when nonbasics becomes superbasic

This controls the extent to which optimization is confined to the current set of basic and superbasic variables (Phase 4 iterations), before one or more nonbasic variables are added to the superbasic set (Phase 3). The parameter \( r \) must be a real number in the range \( 0 < r <= 1 \). When a nonbasic variables \( x_j \) is made superbasic,
the resulting norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be $|Z^T g_0|$. (In fact, the norm will be $|d_j|$, the size of the reduced gradient for the new superbasic variable $x_j$. Subsequent Phase 4 iterations will continue at least until the norm of the reduced-gradient vector satisfies $|[Z^T g_0]| \leq r|Z^T g_0|$ is the size of the largest reduced-gradient component among the superbasic variables.) A smaller value of $r$ is likely to increase the total number of iterations, but may reduce the number of basic changes. A larger value such as $r = 0.9$ may sometimes lead to improved overall efficiency, if the number of superbasic variables has to increase substantially between the starting point and an optimal solution. Other convergence tests on the change in the function being minimized and the change in the variables may prolong Phase 4 iterations. This helps to make the overall performance insensitive to larger values of $r$.

(default = 0.5)

**summary frequency (integer):** Number of iterations between each log line (log file)

A brief form of the iteration log is output to the summary file. In general, one line is output every $i$th minor iteration. In an interactive environment, the output normally appears at the terminal and allows a run to be monitored. If something looks wrong, the run can be manually terminated. The Summary frequency controls summary output in the same as the log frequency controls output to the print file. A value such as `Summary Frequency = 10` or `100` is often adequate to determine if the SOLVE is making progress. If `Print level = 0`, the default value of `Summary Frequency` is 100. If `Print level > 0`, the default value of `Summary Frequency` is 1. If `Print level = 0` and the constraints are nonlinear, the `Summary Frequency` is ignored. Instead, one line is printed at the beginning of each major iteration.

(default = 100)

**superbasics limit (integer):** Maximum number of superbasics

This places a limit on the storage allocated for superbasic variables. Ideally, the parameter $i$ should be set slightly larger than the number of degrees of freedom expected at an optimal solution. For linear problems, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is not more than $m$, the number of general constraints.) The default value of $i$ is therefore 1. For nonlinear problems, the number of degrees of freedom is often called the number of independent variables. Normally, $i$ need not be greater than $n_f + 1$, where $n_f$ is the number of nonlinear variables. For many problems, $i$ may be considerably smaller than $n_f$. This will save storage if $n_f$ is very large. This parameter also sets the Hessian dimension, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, using certain characteristics of the problem.

(default = 1)

**unbounded objective value (real):** Determines when a problem is called unbounded

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

If $|F|$ exceeds the parameter $r$ or if $\alpha$ exceeds the unbounded stepsize, iterations are terminated with the exit message `PROBLEM IS UNBOUNDED (OR BADLY SCALED)`. If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against $r$ can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the `Minor damping parameter`.

(default = $1.0 \times 10^{-20}$)

**unbounded step size (real):** Determines when a problem is called unbounded

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

If $\alpha$ exceeds the parameter $r$ or if $|F|$ exceeds the unbounded objective value, iterations are terminated with the exit message `PROBLEM IS UNBOUNDED (OR BADLY SCALED)`. If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha p)$, before the test against $r$ can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the `Minor damping parameter`.

(default = $1.0 \times 10^{-10}$)
verify constraint gradients (string): Synonym to verify level 2
verify gradients (string): Synonym to verify level 3
verify level (integer): Verification of gradients
  (default = 0)
  0 Cheap test
  1 Check objective
  2 Check Jacobian
  3 Check objective and Jacobian
  ~1 No check
verify no (string): Synonym to verify level 0
verify objective gradients (string): Synonym to verify level 1
verify yes (string): Synonym to verify level 3
weight on linear objective (real): Composite objective weight

The keywords invokes the so-called composite objective technique, if the first solution obtained infeasible, and
if the objective function contains linear terms. While trying to reduce the sum of infeasibilities, the method also
attempts to optimize the linear objective. At each infeasible iteration, the objective function is defined to be

\[ \min_{x} \sigma (c^T x) + \text{sum of infeasibilities} \]

where \( \sigma = 1 \) for minimization and \( \sigma = -1 \) for maximization and \( c \) is the linear objective. If an optimal
solution is reached while still infeasible, \( w \) is reduced by a factor of 10. This helps to allow for the possibility
that the initial \( w \) is too large. It also provides dynamic allowance for the fact the sum of infeasibilities is tending
towards zero. The effect of \( w \) is disabled after five such reductions, or if a feasible solution is obtained. This
option is intended mainly for linear programs. It is unlikely to be helpful if the objective function is nonlinear.
(default = 0.0)

10 Exit Conditions

This section discusses the Exit codes printed by MINOS at the end of the optimization run.

EXIT – Optimal solution found

This is the message we all hope to see! It is certainly preferable to every other message. Of course it is quite
possible that there are model formulation errors, which will (hopefully) lead to unexpected objective values and
solutions. The reported optimum may be a local, and other much better optima may exist.

EXIT – The problem is infeasible

When the constraints are linear, this message can probably be trusted. Feasibility is measured with respect to the
upper and lower bounds on the variables (the bounds on the slack variables correspond to the GAMS constraints).
The message tells us that among all the points satisfying the general constraints \( Ax + s = 0 \), there is apparently
no point that satisfies the bounds on \( x \) and \( s \). Violations as small as the FEASIBILITY TOLERANCE are ignored,
but at least one component of \( x \) or \( s \) violates a bound by more than the tolerance.

Note: Although the objective function is the sum of the infeasibilities, this sum will usually not have been
minimized when MINOS recognizes the situation and exits. There may exist other points that have significantly
lower sum of infeasibilities.

When nonlinear constraints are present, infeasibility is much harder to recognize correctly. Even if a feasible
solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal
with this situation MINOS may relax the bounds on the slacks associated with nonlinear rows. This perturbation
is allowed a fixed number of times. Normally a feasible point will be obtained to the perturbed constraints,
and optimization can continue on the subproblem. However if several consecutive subproblems require such
perturbation, the problem is terminated and declared INFEASIBLE. Clearly this is an ad-hoc procedure. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

EXIT – The problem is unbounded (or badly scaled)

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased by an arbitrary amount without causing a basic variable to violate a bound. A simple way to diagnose such a model is to add an appropriate bound on the objective variable.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the SCALE option.

For nonlinear problems, MINOS monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the UNBOUNDED parameter), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

EXIT – User Interrupt

This exit code is a result of interrupting the optimization process by hitting Ctrl-C. Inside the IDE this is accomplished by hitting the Interrupt button. The solver will finish its current iteration, and return the current solution. This solution can be still intermediate infeasible or intermediate non-optimal.

EXIT – Too many iterations

The iteration limit was hit. Either the ITERLIM, or in some cases the ITERATIONS LIMIT or MAJOR ITERATION LIMIT was too small to solve the problem. In most cases increasing the GAMS ITERLIM option will resolve the problem. In other cases you will need to create a MINOS option file and set a MAJOR ITERATION LIMIT. The listing file will give more information what limit was hit.

The GAMS iteration limit is displayed in the listing file under the section SOLVE SUMMARY. If the ITERLIM was hit, the message will look like:

```
ITERATION COUNT, LIMIT  10001  10000
```

EXIT – Resource Interrupt

The solver hit the RESLIM resource limit, which is a time limit. It returned the solution at that time, which may be still intermediate infeasible or intermediate non-optimal.

The GAMS resource limit is displayed in the listing file under the section SOLVE SUMMARY. If the GAMS RESLIM was hit, the message will look like:

```
RESOURCE USAGE, LIMIT  1001.570  1000.000
```

EXIT – The objective has not changed for many iterations

This is an emergency measure for the rare occasions when the solution procedure appears to be cycling. Suppose that a zero step is taken for several consecutive iterations, with a basis change occurring each time. It is theoretically possible for the set of basic variables to become the same as they were one or more iterations earlier. The same sequence of iterations would then occur ad infinitum.

EXIT – The Superbasics Limit is too small

The problem appears to be more non-linear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and it is needed to increase the number of superbasics. You can use the option SUPERBASICS LIMIT to increase the limit. See also option HESSIAN DIMENSION.

EXIT – Constraint and objective function could not be calculated

The function or gradient could not be evaluated. This means the algorithm tried to take a log of a negative number, a square root of a negative number or there was an expression $x^2$ with $x \leq 0$ or something related like a floating point overflow. The listing file will contain an indication in which equation this happened. The solution is to add bounds so that all functions can be properly evaluated. E.g. if you have an expression $x^2$, then add a bound X.LO=0.001;.
In many cases the algorithm can recover from function evaluation errors, for instance if they happen in the line search. In this case the algorithm can not recover, and requires a reliable function or gradient evaluation.

EXIT – Function evaluation error limit
The limit of allowed function evaluation errors DOMLIM has been exceeded.
This means the algorithm tried too many time to take a log of a negative number, a square root of a negative number or there was an expression $x^y$ with $x \leq 0$ or something related like a floating point overflow. The listing file will contain an indication in which equation this happened.

The simple way to solve this is to increase the GAMS DOMLIM setting, but in general it is better to add bounds. E.g. if you have an expression $x^y$, then add a bound $x.LO=0.001$.

EXIT – The current point can not be improved
The line search failed. This can happen if the model is very nonlinear or if the functions are nonsmooth (using a DNLP model type).
If the model is non-smooth, consider a smooth approximation. It may be useful to check the scaling of the model and think more carefully about choosing a good starting point. Sometimes it can help to restart the model with full scaling turned on:

```gams
option nlp=minos;
solve m minimizing z using nlp; // this one gives "current point cannot be improved"
file fopt /minos.opt/; // write option file
put fopt;
put "scale all variables"/;
putclose;
m.optfile=1;
solve m minimizing z using nlp; // solve with "scale all variables"
```

EXIT – Numerical error in trying to satisfy the linear constraints (or the linearized constraints)
The basis is very ill-conditioned.
This is often a scaling problem. Try full scaling, or better: use user-defined scaling using the GAMS scaling syntax.

EXIT – Not enough storage to solve the model
The estimate of the workspace needed to solve the model has turned out to be insufficient. You may want to increase the workspace by using the GAMS WORK option, or the M.WORKSPACE model suffix.
The listing file and log file (screen) will contain a message of the currently allocated workspace. This gives an indication of how much you should allocate, e.g. 1.5 times as much.

EXIT– Systems error
This is a catch all return for other serious problems. Check the listing file for more messages. If needed rerun the model with OPTION SYSOUT=ON.

Bibliography


1 Introduction

MOSEK is a software package for the solution of linear, mixed-integer linear, quadratic, mixed-integer quadratic, quadratically constraint, and convex nonlinear mathematical optimization problems. MOSEK is particularly well suited for solving large-scale linear, convex quadratically constraint, and conic quadratic programs using an extremely efficient interior point algorithm.
These problem classes can be solved using an appropriate optimizer built into MOSEK. All the optimizers available in MOSEK are built for the solution of large-scale sparse problems. Current optimizers include:

- Interior-point optimizer for all continuous problems
- Conic interior-point optimizer for conic quadratic problems
- Simplex optimizer for linear problems
- Mixed-integer optimizer based on a branch and cut technology
- Conic Mixed-integer optimizer based on a branch and cut technology

1.1 Licensing

Licensing of GAMS/MOSEK is similar to other GAMS solvers. MOSEK is licensed in three different ways:

- **GAMS/MOSEK Base:**
  All model types. Using MOSEK’s conic mixed-integer optimizer for the solution of models involving discrete variables.

- **GAMS/MOSEK Extended:**
  Same as GAMS/MOSEK Base, but using the mixed-integer optimizer for the solution of models involving discrete variables by default

- **GAMS/MOSEK Solver Link:**
  Users must have a separate, licensed MOSEK system. For users who wish to use MOSEK within GAMS and also in other environments.

- **GAMS/OsiMosek:**
  Finally, a bare-bone interface to the LP and MIP solver of Mosek is available under the name OSIMOSEK. It comes free of charge with any GAMS system.

For more information contact sales@gams.com. For information regarding MOSEK standalone or interfacing MOSEK with other applications contact sales@mosek.com.

1.2 Reporting of Infeasible/Unbounded Models

MOSEK determines if either the primal or the dual problem is infeasible by means of a Farkas certificate. In such a case MOSEK returns a certificate indicating primal or dual infeasibility. A primal infeasibility certificate indicates a primal infeasible model and the certificate is reported in the marginals of the equations in the listing file. The primal infeasibility certificate for a minimization problem

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Ax = b, \\
& \quad x \geq 0
\end{align*}
\]

is the solution \( y \) satisfying:

\[
A^Ty \leq 0, \quad b^Ty > 0
\]

Since GAMS reports all model statuses in the primal space, the notion of dual infeasibility does not exist and GAMS reports a status of unbounded, which assumes the primal problem is feasible. Although GAMS reports the primal as unbounded, there is the possibility that both the primal and dual problem are infeasible. To check if this is the case, the user can set appropriate upper and lower bounds on the objective variable, using the \( (\text{variable}).\text{LO} \) and \( (\text{variable}).\text{UP} \) suffixes and resolve.

For more details on primal and dual infeasibility certificates see the MOSEK User’s manual at www.mosek.com.
1.3 Solving Problems in Parallel

If a computer has multiple CPUs (or a CPU with multiple cores), then it might be advantageous to use the multiple CPUs to solve the optimization problem. For instance if you have two CPUs you may want to exploit the two CPUs to solve the problem in the half time. MOSEK can exploit multiple CPUs.

Parallelized Optimizers

MOSEK’s interior-point and the conic mixed-integer optimizer have been parallelized.

This implies that whenever the MOSEK interior-point optimizer should solve an optimization problem, then it will try to divide the work so each CPU gets a share of the work. The user decides how many CPUs MOSEK should exploit. Unfortunately, it is not always easy to divide the work. Also some of the coordination work must occur in sequential. Therefore, the speed-up obtained when using multiple CPUs is highly problem dependent. However, as a rule of thumb if the problem solves very quickly i.e. in less than 60 seconds, then it is not advantageous of using the parallel option.

The parameter $\text{MSK\_IPAR\_NUM\_THREADS}$ sets the number of threads (and therefore the number of CPU’s) that the interior point optimizer will use.

Concurrent Optimizer

An alternative to use a parallelized optimizer is the concurrent optimizer. The idea of the concurrent optimizer is to run multiple optimizers on the same problem concurrently. For instance the interior-point and the dual simplex optimizers may be applied to an linear optimization problem concurrently. The concurrent optimizer terminates when the first optimizer has completed and reports the solution of the fastest optimizer. That way a new optimizer has been created which essentially has the best performance of the interior-point and the dual simplex optimizer.

Hence, the concurrent optimizer is the best one to use if there multiple optimizers available in MOSEK for the problem and you cannot say beforehand which one is the best one. For more details inspect the $\text{MSK\_IPAR\_CONCURRENT\_\ast}$ options.

1.4 The Infeasibility Report

MOSEK has some facilities for diagnosing the cause of a primal or dual infeasibility. They can be turned on using the parameter setting $\text{MOSEK\_MSK\_IPAR\_INFEAS\_REPORT\_AUTO}$. This causes MOSEK to print a report about an infeasible subset of the constraints, when an infeasibility is encountered. Moreover, the parameter $\text{MOSEK\_MSK\_IPAR\_INFEAS\_REPORT\_LEVEL}$ controls the amount info presented in the infeasibility report. We will use the transport.gms example from the GAMS Model Library with increased demand $(b(j) = 1.6 * b(j))$ to make the model infeasible. MOSEK produces the following infeasibility report

MOSEK PRIMAL INFEASIBILITY REPORT.

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Dual lower</th>
<th>Dual upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>supply(seattle)</td>
<td>none</td>
<td>3.500000e+02</td>
<td>0.000000e+00</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>2</td>
<td>supply(san-diego)</td>
<td>none</td>
<td>6.000000e+02</td>
<td>0.000000e+00</td>
<td>1.000000e+00</td>
</tr>
<tr>
<td>3</td>
<td>demand(new-york)</td>
<td>5.200000e+02</td>
<td>none</td>
<td>1.000000e+00</td>
<td>0.000000e+00</td>
</tr>
<tr>
<td>4</td>
<td>demand(chicago)</td>
<td>4.800000e+02</td>
<td>none</td>
<td>0.000000e+00</td>
<td>0.000000e+00</td>
</tr>
<tr>
<td>5</td>
<td>demand(topeka)</td>
<td>4.400000e+02</td>
<td>none</td>
<td>1.000000e+00</td>
<td>0.000000e+00</td>
</tr>
</tbody>
</table>

The following bound constraints are involved in the infeasibility.

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Dual lower</th>
<th>Dual upper</th>
</tr>
</thead>
</table>


which indicates which constraints and bounds that are important for the infeasibility i.e.~causing the infeasibility. The infeasibility report is divided into two sections where the first section shows which constraints that are important for the infeasibility. In this case the important constraints are supply and demand. The values in the columns Dual lower and Dual upper are also useful, because if the dual lower value is different from zero for a constraint, then it implies that the lower bound on the constraint is important for the infeasibility. Similarly, if the dual upper value is different from zero on a constraint, then this implies the upper bound on the constraint is important for infeasibility.

1.5 Nonlinear Programs

MOSEK can efficiently solve convex programs, but is not intended for nonconvex optimization. For nonconvex programs, MOSEK can detect some nonconvexities and will print out a warning message and terminate. If MOSEK does not detect nonconvexities for a nonconvex model, the optimizer may continue but stagnate. Hence care must be taken when solving nonlinear programs if convexity is not immediately known.

1.6 Modeling Issues Involving Convex Programs

It is often preferable to model convex programs in separable form, if it is possible. Consider the following example of minimizing an objective function \( f(x) \):

\[
  f(x) = \log(a^*x)
\]

where \( a \in \mathbb{R}^n \) is a parameter and \( x \in \mathbb{R}^n \) the decision variable. The equation implies an implicit constraint of \( a^*x > 0 \). Unfortunately, domain violations can still occur because no restrictions are set on \( a^*x \). A better approach is to introduce an intermediate variable \( y \):

\[
  f(x) = \log(y) \\
  y = a^*x \\
  y \geq 0
\]

This accomplishes two things. It implies an explicit bound on \( a^*x \), thereby reducing the risk of domain violations. Secondly, it speeds up computation since computations of gradients and Hessians in the first (non-separable) form are more expensive. Finally, it reduces the amount of memory needed (see the section on "Memory Options")

2 Quadratic Conic Programming

MOSEK is well suited for solving generalized linear programs involving nonlinear conic constraints. Conic programming is useful in a wide variety of application areas \(^1\) including engineering and financial management. Conic programming has been used, for example, in antenna array weight design, grasping force optimization, finite impulse response (FIR) filter design, and portfolio optimization. The Mosek web site (http://mosek.com/resources/doc) has an excellent modeling manual for conic problems.

This section gives an overview of quadratic conic programming and how these conic constraints are implemented in GAMS.

2.1 Introduction

Conic programs can be thought of as generalized linear programs with the additional nonlinear constraint \( x \in C \), where \( C \) is required to be a convex cone. The resulting class of problems is known as conic optimization and has the following form:

\[
  \begin{align*}
    \text{minimize} & & c^T x \\
    \text{subject to} & & Ax \leq r', \\
                       & & x \in [l', u'] \\
                       & & x \in C
  \end{align*}
\]

where \( A \in \mathbb{R}^{m \times n} \) is the constraint matrix, \( x \in \mathbb{R}^{n} \) the decision variable, and \( c \in \mathbb{R}^{n} \) the objective function cost coefficients. The vector \( r^c \in \mathbb{R}^{m} \) represents the right hand side and the vectors \( l^t, u^t \in \mathbb{R}^{n} \) are lower and upper bounds on the decision variable \( x \).

Now partition the set of decision variables \( x \) into sets \( S^t, t = 1, ..., k \), such that each decision variables \( x \) is a member of at most one set \( S^t \). For example, we could have

\[
S^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \quad \text{and} \quad S^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.
\]

Let \( x^{S^t} \) denote the variables \( x \) belonging to set \( S^t \). Then define

\[
C := \{ x \in \mathbb{R}^{n} : x^{S^t} \in C_t, t = 1, ..., k \}
\]

where \( C_t \) must have one of the following forms:

- Quadratic cone: (also referred to as Lorentz or ice cream cone)

\[
C_t = \{ x \in \mathbb{R}^{n} : x_1 \geq \sqrt{\sum_{j=2}^{n} x_j^2} \}.
\]

- Rotated quadratic cone: (also referred to as hyperbolic constraints)

\[
C_t = \{ x \in \mathbb{R}^{n} : 2x_1x_2 \geq \sum_{j=3}^{n} x_j^2, x_1, x_2 \geq 0 \}.
\]

These two types of cones allow the formulation of quadratic, quadratically constrained, and many other classes of nonlinear convex optimization problems.

### 2.2 Implementation of Quadratic Conic Constraints in GAMS

GAMS handles conic equations using the \( =C= \) equation type. The conic cases are written as:

- Quadratic cone:

\[
x('1') \ =C= \ \text{sum(i\[\{not sameas(i,'1')\}], x(i));}
\]

- Rotated quadratic cone:

\[
x('1')+x('2') \ =C= \ \text{sum(i\[\{not sameas(i,'1') \text{ and not sameas(i,'2')\}], x(i));}
\]

Note that the resulting nonlinear conic constraints result in "linear" constraints in GAMS. Thus the original nonlinear formulation is in fact a linear model in GAMS. We remark that we could formulate conic problems as regular NLP using constraints:

- Quadratic cone:

\[
x('1') \ =G= \ \text{sqrt[ \text{sum(i\[\{not sameas(i,'1')\}], sqr(x(i))])];}
\]

- Rotated quadratic cone:

\[
2*x('1')*x('2') \ =G= \ \text{sum(i\[\{not sameas(i,'1') \text{ and not sameas(i,'2')\}], sqr(x(i))};
\]

where 'x('1')' and 'x('2')' are positive variables

The example below illustrates the different formulations for conic programming problems. Note that the conic optimizer in MOSEK usually outperforms a general NLP method for the reformulated (NLP) cone problems.
2.3 Example

Consider the following example (cone2.gms) which illustrates the use of rotated conic constraints. We will give reformulations of the original problem in regular NLP form using conic constraints and in conic form.

The original problem is:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i} d_i x_i \\
\text{subject to} & \quad d^T x \leq b, \quad x_i \in [l_i, u_i], \quad l_i > 0, \quad d_i \geq 0, \quad i = 1, 2, \ldots, n
\end{align*}
\]  

(5)

where \( x \in \mathbb{R}^n \) is the decision variable, \( d, a, l, u \in \mathbb{R}^n \) parameters, and \( b \in \mathbb{R} \) a scalar parameter. The original model (5) can be written in GAMS using the equations:

```gams
1 defobj.. sum(n, d(n)/x(n)) =E= obj;
2 e1.. sum(n, a(n)*x(n)) =L= b;
3 Model orig /defobj, e1/;
4 x.lo(n) = l(n);
5 x.up(n) = u(n);
```

We can write an equivalent NLP formulation, replacing the objective function and adding another constraint:

\[
\begin{align*}
\text{subject to} & \quad d^T x \leq b \\
& \quad 2t_i x_i \geq 2, \quad i = 1, \ldots, n \\
& \quad x \in [l, u], \quad l > 0, \quad d_i \geq 0
\end{align*}
\]  

(6)

where \( t \in \mathbb{R}^n \) is a new decision variable. The GAMS formulation of this NLP (model cnlp) is:

```gams
1 defobjc.. sum(n, d(n)*t(n)) =E= obj;
2 e1.. sum(n, a(n)*x(n)) =L= b;
3 conenlp(n).. 2*t(n)*x(n) =G= 2;
4 Model cnlp /defobjc, e1, conenlp/;
5 x.lo(n) = l(n);
6 x.up(n) = u(n);
```

We can change the equality to an inequality since the parameter \( d_i \geq 0 \) and we are dealing with a minimization problem. Also, note that the constraint `conenlp(n)` is almost in rotated conic form. If we introduce a variable \( z \in \mathbb{R}^n \), \( z_i = \sqrt{2} \), then we can reformulate the problem using conic constraints as:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i} d_i t_i \\
\text{subject to} & \quad d^T x \leq b \\
& \quad z_i = \sqrt{2} \\
& \quad 2t_i x_i \geq z_i^2, \quad i = 1, \ldots, n \\
& \quad x \in [l, u], \quad l > 0, \quad d_i \geq 0
\end{align*}
\]  

(7)

The GAMS formulation using conic equations =C= is:

```gams
1 defobjc.. sum(n, d(n)*t(n)) =E= obj;
2 e1.. sum(n, a(n)*x(n)) =L= b;
3 e2(n).. z(n) =E= sqrt(2);
4 cone(n).. x(n) + t(n) =C= z(n);
5 Model clp /defobjc, e1, e2, cone/;
6 x.lo(n) = l(n);
7 x.up(n) = u(n);
```

Note that this formulation is a linear program in GAMS, although the constraints `cone(n)` represent the nonlinear rotated quadratic cone constraint.

The complete model is listed below:

```gams
1 Set n / n1*n10 /;
2 Parameter d(n), a(n), l(n), u(n);
3 Scalar b;
4 5 d(n) = uniform(1,2);
```
3 The MOSEK Options

MOSEK works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are reslim, nodlim, optca, optcr, and optfile. The option iterlim works only for the simplex optimizer. A description of all available GAMS options can be found in Chapter "Using Solver Specific Options".

We remark that MOSEK contains many complex solver options, many of which require a deep understanding of the algorithms used. For a complete description of the many MOSEK options, consult the MOSEK User’s Guide, available online at www.mosek.com.

If you specify <modelname>.optfile = 1; before the SOLVE statement in your GAMS model, MOSEK will then look for and read an option file with the name mosek.opt (see "Using Solver Specific Options" for general use of solver option files). The syntax for the MOSEK option file is

    optname value

with one option on each line.

For example,

    MSK_IPAR_INTPNT_MAX_ITERATIONS 20
    MSK_IPAR_INTPNT_SCALING MSK_SCALING_NONE

The first option specifies the maximum number of interior-point iterations, in this case 20. The second option indicates that no scaling should be used.

4 Summary of MOSEK Options

4.1 General and Preprocessing Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_ANA_SOL_INFEAS_TOL</td>
<td>Report constraint bound violation if it exceeds this tolerance.</td>
<td>1e-6</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>MSK_DPAR_BASIS_REL_TOL_S</td>
<td>Maximum relative dual bound violation allowed in an optimal basic solution.</td>
<td>1.0e-12</td>
</tr>
<tr>
<td>MSK_DPAR_BASIS_TOL_S</td>
<td>Maximum absolute dual bound violation in an optimal basic solution.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>MSK_DPAR_BASIS_TOL_X</td>
<td>Maximum absolute primal bound violation allowed in an optimal basic solution.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>MSK_DPAR_CHECK_CONVEXITY_REL_TOL</td>
<td>This parameter controls when the full convexity check declares a problem to be non-convex.</td>
<td>1e-10</td>
</tr>
<tr>
<td>MSK_DPAR_OPTIMIZER_MAX_TIME</td>
<td>Maximum amount of time the optimizer is allowed to spent on the optimization.</td>
<td>GAMS ResLim</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP</td>
<td>Absolute tolerance employed by the linear dependency checker.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_AIJ</td>
<td>Absolute zero tolerance employed for $a_{ij}$ in the presolve.</td>
<td>1.0e-12</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_REL_LINDEP</td>
<td>Relative tolerance employed by the linear dependency checker.</td>
<td>1.0e-10</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_S</td>
<td>Absolute zero tolerance employed for $s_i$ in the presolve.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>MSK_DPAR_PRESOLVE_TOL_X</td>
<td>Absolute zero tolerance employed for $x_j$ in the presolve.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>MSK_IPAR_AUTO_SORT_A_BEFORE_OPT</td>
<td>Controls whether the elements in each column of $A$ are sorted before an optimization is performed.</td>
<td>MSK_OFF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS</td>
<td>The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.</td>
<td>2</td>
</tr>
<tr>
<td>MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX</td>
<td>Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.</td>
<td>2</td>
</tr>
<tr>
<td>MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX</td>
<td>Priority of the free simplex optimizer when selecting solvers for concurrent optimization.</td>
<td>3</td>
</tr>
<tr>
<td>MSK_IPAR_CONCURRENT_PRIORITY_INTPNT</td>
<td>Priority of the interior-point algorithm when selecting solvers for concurrent optimization.</td>
<td>4</td>
</tr>
<tr>
<td>MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX</td>
<td>Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_INFEAS_REPORT_AUTO</td>
<td>Controls infeasibility report production.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_INFEAS_REPORT_LEVEL</td>
<td>Controls the amount of information presented in an infeasibility report.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_NUM_THREADS</td>
<td>Controls the number of threads employed by the optimizer.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_OPTIMIZER</td>
<td>The parameter controls which optimizer is used to optimize the task.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES</td>
<td>Control the maximum number of times the eliminator is tried.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_ELIMINATOR_USE</td>
<td>Controls whether free or implied free variables are eliminated from the problem.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_ELIM_FILL</td>
<td>Controls the maximum amount of fill-in in the elimination phase of the presolve.</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_LEVEL</td>
<td>Currently not used.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH</td>
<td>Controls the linear dependency check which is potentially computationally expensive.</td>
<td>100</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH</td>
<td>Controls the linear dependency check which is potentially computationally expensive.</td>
<td>100</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_LINDEP_USE</td>
<td>Controls whether the linear constraints are checked for linear dependencies.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS</td>
<td>Controls the maximum number reductions performed by the presolve.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_PRESOLVE_USE</td>
<td>Controls whether the presolve is applied to a problem before it is optimized.</td>
<td>MSK_PRESOLVE_MODE_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER</td>
<td>Controls which optimizer that is used to find the optimal repair.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td>MSK_SPAR_DATA_FILE_NAME</td>
<td>Problem data is written to this file. File extension determines format.</td>
<td></td>
</tr>
<tr>
<td>MSK_SPAR_PARAM_READ_FILE_NAME</td>
<td>Modifications to the parameter database is read from this file.</td>
<td></td>
</tr>
<tr>
<td>MSK_SPAR_PARAM_WRITE_FILE_NAME</td>
<td>The parameter database is written to this file.</td>
<td></td>
</tr>
</tbody>
</table>

### 4.2 Problem Data Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_DPAR_DATA_TOL_AIJ</td>
<td>Absolute zero tolerance for elements in $A$.</td>
<td>$1.0 \times 10^{-12}$</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_AIJ_HUGE</td>
<td>An element in $A$ which is larger than this value in absolute size causes an error.</td>
<td>$1.0 \times 10^20$</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_AIJ_LARGE</td>
<td>An element in $A$ which is larger than this value in absolute size causes a warning message.</td>
<td>$1.0 \times 10^10$</td>
</tr>
<tr>
<td>MSK_DPAR_DATA_TOL_BOUND_INF</td>
<td>Any bound which in absolute value is greater than this parameter is considered infinite.</td>
<td>$1.0 \times 10^16$</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>MSK_DPAR_DATA_TOL_BOUND_WRN</strong></td>
<td>If a bound value is larger than this value in absolute size, then a warning message is issued.</td>
<td>1.0e8</td>
</tr>
<tr>
<td><strong>MSK_DPAR_DATA_TOL_CJ_LARGE</strong></td>
<td>An element in $c$ which is larger than this value in absolute terms causes a warning message.</td>
<td>1.0e8</td>
</tr>
<tr>
<td><strong>MSK_DPAR_DATA_TOL_C_HUGE</strong></td>
<td>An element in $c$ larger than the value of this parameter in absolute terms generates an error.</td>
<td>1.0e16</td>
</tr>
<tr>
<td><strong>MSK_DPAR_DATA_TOL_QIJ</strong></td>
<td>Absolute zero tolerance for elements in $Q$ matrices.</td>
<td>1.0e-16</td>
</tr>
<tr>
<td><strong>MSK_DPAR_DATA_TOL_X</strong></td>
<td>Zero tolerance for constraints and variables.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td><strong>MSK_DPAR_LOWER_OBJ_CUT</strong></td>
<td>Lower objective limit.</td>
<td>-1.0e30</td>
</tr>
<tr>
<td><strong>MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH</strong></td>
<td>Lower objective limit threshold.</td>
<td>-0.5e30</td>
</tr>
<tr>
<td><strong>MSK_DPAR_UPPER_OBJ_CUT</strong></td>
<td>Upper objective limit.</td>
<td>1.0e30</td>
</tr>
<tr>
<td><strong>MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH</strong></td>
<td>Upper objective limit threshold.</td>
<td>0.5e30</td>
</tr>
<tr>
<td><strong>MSK_IPAR_CHECK_CONVEXITY</strong></td>
<td>Specify the level of convexity check on quadratic problems</td>
<td><strong>MSK_CHECK_CONVEXITY_FULL</strong></td>
</tr>
</tbody>
</table>

### 4.3 Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK_IPAR_LOG</strong></td>
<td>Controls the amount of log information.</td>
<td>10</td>
</tr>
<tr>
<td><strong>MSK_IPAR_LOG_BI</strong></td>
<td>Controls the amount of output printed by the basis identification procedure.</td>
<td>4</td>
</tr>
<tr>
<td><strong>MSK_IPAR_LOG_BI_FREQ</strong></td>
<td>Controls logging frequency.</td>
<td>2500</td>
</tr>
<tr>
<td><strong>MSK_IPAR_LOG_CHECK_CONVEXITY</strong></td>
<td>Controls logging in convexity check on quadratic problems.</td>
<td>0</td>
</tr>
<tr>
<td><strong>MSK_IPAR_LOG_CONCURRENT</strong></td>
<td>Controls amount of output printed by the concurrent optimizer.</td>
<td>1</td>
</tr>
<tr>
<td><strong>MSK_IPAR_LOG_FACTOR</strong></td>
<td>If turned on, then the factor log lines are added to the log.</td>
<td>1</td>
</tr>
<tr>
<td><strong>MSK_IPAR_LOG_FEAS_REPAIR</strong></td>
<td>Controls the amount of output printed when performing feasibility repair.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_HEAD</td>
<td>If turned on, then a header line is added to the log.</td>
<td>1</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_INFEAS_ANA</td>
<td>Controls amount of output printed by the infeasibility analyzer procedures.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_INTPNT</td>
<td>Controls amount of output printed by the interior-point optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_MIO</td>
<td>Controls the log level for the mixed-integer optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_MIO_FREQ</td>
<td>Controls how frequent the mixed-integer optimizer prints the log line.</td>
<td>1000</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_NONCONVEX</td>
<td>Controls amount of output printed by the nonconvex optimizer.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_OPTIMIZER</td>
<td>Controls the amount of general optimizer information that is logged.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_ORDER</td>
<td>If turned on, then factor lines are added to the log.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_PARAM</td>
<td>Controls the amount of information printed out about parameter changes.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_PRESOLVE</td>
<td>Controls amount of output printed by the presolve procedure.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_RESPONSE</td>
<td>Controls amount of output printed when response codes are reported.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_LOG_SENSITIVITY</td>
<td>Controls the amount of logging during the sensitivity analysis.</td>
<td>1</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>MSK_IPAR_LOG_SENSITIVITY_OPT</code></td>
<td>Controls the amount of logging from the optimizers employed during the sensitivity analysis.</td>
<td>0</td>
</tr>
<tr>
<td><code>MSK_IPAR_LOG_SIM</code></td>
<td>Controls amount of output printed by the simplex optimizer.</td>
<td>4</td>
</tr>
<tr>
<td><code>MSK_IPAR_LOG_SIM_FREQ</code></td>
<td>Controls simplex logging frequency.</td>
<td>1000</td>
</tr>
<tr>
<td><code>MSK_IPAR_LOG_STORAGE</code></td>
<td>When turned on, MOSEK prints messages regarding the storage usage and allocation.</td>
<td>0</td>
</tr>
<tr>
<td><code>MSK_IPAR_MAX_NUM_WARNINGS</code></td>
<td>Warning level. A higher value results in more warnings.</td>
<td>10</td>
</tr>
<tr>
<td><code>MSK_IPAR_WARNING_LEVEL</code></td>
<td>Warning level. A higher value implies more warnings.</td>
<td>1</td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS</code></td>
<td>Controls if the writer ignores incompatible conic items.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS</code></td>
<td>Controls if the writer ignores incompatible problem items when writing files.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS</code></td>
<td>Controls if the writer ignores general non-linear terms or produces an error.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS</code></td>
<td>Controls if the writer ignores incompatible psd items.</td>
<td><code>MSK_OFF</code></td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_LP_LINE_WIDTH</code></td>
<td>Maximum width of line in an LP file written by MOSEK.</td>
<td>80</td>
</tr>
<tr>
<td><code>MSK_IPAR_WRITE_LP_QUOTED_NAMES</code></td>
<td>If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.</td>
<td><code>MSK_ON</code></td>
</tr>
</tbody>
</table>
### 4.4 Interior Point Optimizer Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSK_DPAR_FEASREPAIR_TOL</code></td>
<td>Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.</td>
<td>1.0e-10</td>
</tr>
<tr>
<td><code>MSK_DPAR_INTPNT_CO_TOL_DFEAS</code></td>
<td>Dual feasibility tolerance used by the conic interior-point optimizer.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td><code>MSK_DPAR_INTPNT_CO_TOL_INFEAS</code></td>
<td>Controls when the conic interior-point optimizer declares the model primal or dual infeasible.</td>
<td>1.0e-10</td>
</tr>
<tr>
<td><code>MSK_DPAR_INTPNT_CO_TOL_MU_RED</code></td>
<td>Optimality tolerance for the conic solver.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td><code>MSK_DPAR_INTPNT_CO_TOL_NEAR_REL</code></td>
<td>Termination tolerance multiplier that is used if no accurate solution can be found.</td>
<td>1000</td>
</tr>
<tr>
<td><code>MSK_DPAR_INTPNT_CO_TOL_PFEAS</code></td>
<td>Primal feasibility tolerance used by the conic interior-point optimizer.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>Environment</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_CO_TOL_REL_GAP</td>
<td>Relative gap termination tolerance used by the conic interior-point optimizer.</td>
<td>1.0e-7</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_NL_MERIT_BAL</td>
<td>Controls if the complementarity and infeasibility is converging to zero at about equal rates.</td>
<td>1.0e-4</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_NL_TOL_DFEAS</td>
<td>Dual feasibility tolerance used when a nonlinear model is solved.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_NL_TOL_MU_RED</td>
<td>Relative complementarity gap tolerance.</td>
<td>1.0e-12</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_NL_TOL_NEAR_REL</td>
<td>Termination tolerance multiplier that is used if no accurate solution can be found.</td>
<td>1000.0</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_NL_TOL_PFEAS</td>
<td>Primal feasibility tolerance used when a nonlinear model is solved.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_NL_TOL_REL_GAP</td>
<td>Relative gap termination tolerance for nonlinear problems.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_NL_TOL_REL_STEP</td>
<td>Relative step size to the boundary for general nonlinear optimization problems.</td>
<td>0.995</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_DFEAS</td>
<td>Dual feasibility tolerance used for linear and quadratic optimization problems.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_DSAFE</td>
<td>Controls the initial dual starting point used by the interior-point optimizer.</td>
<td>1.0</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_INFEAS</td>
<td>Controls when the optimizer declares the model primal or dual infeasible.</td>
<td>1.0e-10</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_MU_RED</td>
<td>Relative complementarity gap tolerance.</td>
<td>1.0e-16</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_PATH</td>
<td>Controls how close the interior-point optimizer follows the central path.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_PFEAS</td>
<td>Primal feasibility tolerance used for linear and quadratic optimization problems.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_PSAFE</td>
<td>Controls the initial primal starting point used by the interior-point optimizer.</td>
<td>1.0</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_REL_GAP</td>
<td>Relative gap termination tolerance.</td>
<td>1.0e-8</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_REL_STEP</td>
<td>Relative step size to the boundary for linear and quadratic optimization problems.</td>
<td>0.9999</td>
</tr>
<tr>
<td>MSK_DPAR_INTPNT_TOL_STEP_SIZE</td>
<td>Step size tolerance value.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_BASIS</td>
<td>Controls whether the interior-point optimizer also computes an optimal basis.</td>
<td>MSK_BI_ALWAYS</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_DIFF_STEP</td>
<td>Controls whether different step sizes are allowed in the primal and dual space.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_FACTOR_DEBUG_LVL</td>
<td>Controls factorization debug level.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_FACTOR_METHOD</td>
<td>Controls the method used to factor the Newton equation system.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_MAX_ITERATIONS</td>
<td>Controls the maximum number of iterations allowed in the interior-point optimizer.</td>
<td>400</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_MAX_NUM_COR</td>
<td>Controls the maximum number of correctors allowed by the multiple corrector procedure.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS</td>
<td>Maximum number of steps to be used by the iterative refinement of the search direction.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_OFF_COL_TRH</td>
<td>Controls how many offending columns are detected in the Jacobian of the constraint matrix.</td>
<td>40</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_ORDER_METHOD</td>
<td>Controls the ordering strategy.</td>
<td>MSK_ORDER_METHOD_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_REGULARIZATION_USE</td>
<td>Controls whether regularization is allowed.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_SCALING</td>
<td>Controls how the problem is scaled before the interior-point optimizer is used.</td>
<td>MSK_SCALING_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_INTPNT_SOLVE_FORM</td>
<td>Controls whether the primal or the dual problem is solved.</td>
<td>MSK_SOLVE_FREE</td>
</tr>
</tbody>
</table>
### 4.5 Simplex Optimizer and Basis Identification Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK.DPAR_SIMPLEX_ABS_TOL_PIV</strong></td>
<td>Absolute pivot tolerance employed by the simplex optimizers.</td>
<td>1.0e-7</td>
</tr>
<tr>
<td><strong>MSK.DPAR_SIM_LU_TOL_REL_PIV</strong></td>
<td>Relative pivot tolerance for LU factorization in simplex and basis identification.</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>MSK.IPAR_BI_CLEAN_OPTIMIZER</strong></td>
<td>Controls which optimizer is used in the clean-up phase.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td><strong>MSK.IPAR_BI_IGNORE_MAX_ITER</strong></td>
<td>Controls if basis identification is performed under certain conditions.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td><strong>MSK.IPAR_BI_IGNORE_NUM_ERROR</strong></td>
<td>Turns on basis identification if interior-point optimizer is terminated due to a numerical problem.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td><strong>MSK.IPAR_BI_MAX_ITERATIONS</strong></td>
<td>Maximum number of iterations after basis identification.</td>
<td>1000000</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_BASIS_FACTOR_USE</strong></td>
<td>Controls whether a (LU) factorization of the basis is used in a hot-start.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_DEGEN</strong></td>
<td>Controls how aggressively degeneration is handled.</td>
<td>MSK_SIM_DEGEN_FREE</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_DUAL_CRASH</strong></td>
<td>Controls whether crashing is performed in the dual simplex optimizer.</td>
<td>GAMS BRatio</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_DUAL.Restrict_Selection</strong></td>
<td>Controls how aggressively restricted selection is used.</td>
<td>50</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_DUAL_SELECTION</strong></td>
<td>Controls the dual simplex strategy.</td>
<td>MSK_SIM_SELECTION_FREE</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_EXPLOIT_DUPVEC</strong></td>
<td>Controls if the simplex optimizers are allowed to exploit duplicated columns.</td>
<td>MSK_SIM_EXPLOIT_DUPVEC_OFF</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_HOTSTART</strong></td>
<td>Controls the type of hot-start that the simplex optimizer perform.</td>
<td>MSK_SIM_HOTSTART_FREE</td>
</tr>
<tr>
<td><strong>MSK.IPAR_SIM_HOTSTART_LU</strong></td>
<td>Determines if the simplex optimizer should exploit the initial factorization.</td>
<td>MSK_ON</td>
</tr>
</tbody>
</table>

**MSK.IPAR_INTPNT_STARTING_POINT**

Starting point used by the interior-point optimizer.

**MSK_STARTING_POINT_FREE**
### Mixed Integer Optimizer Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK_IPAR_SIM_MAX_ITERATIONS</td>
<td>Maximum number of iterations that can be used by a simplex optimizer.</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_MAX_NUM_SETBACKS</td>
<td>Controls how many set-backs are allowed within a simplex optimizer.</td>
<td>250</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_NON_SINGULAR</td>
<td>Controls if the simplex optimizer ensures a non-singular basis, if possible.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_PRIMAL_CRASH</td>
<td>Controls whether crashing is performed in the primal simplex optimizer.</td>
<td>GAMS BRatio</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION</td>
<td>Controls how aggressively restricted selection is used.</td>
<td>50</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_PRIMAL_SELECTION</td>
<td>Controls the primal simplex strategy.</td>
<td>MSK_SIM_SELECTION_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_REFACTOR_FREQ</td>
<td>Controls how frequent the basis is refactorized.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_REFORMULATION</td>
<td>Controls if the simplex optimizers are allowed to reformulate the problem.</td>
<td>MSK_SIM_REFORMULATION_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SAVE_LU</td>
<td>Controls storage of LU factorization.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SCALING</td>
<td>Controls how much effort is used in scaling the problem before a simplex optimizer is used.</td>
<td>MSK_SCALING_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SCALING_METHOD</td>
<td>Controls how the problem is scaled before a simplex optimizer is used.</td>
<td>MSK_SCALING_METHOD_POW2</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SOLVE_FORM</td>
<td>Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.</td>
<td>MSK_SOLVE_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_STABILITY_PRIORITY</td>
<td>Controls how high priority the numerical stability should be given.</td>
<td>50</td>
</tr>
<tr>
<td>MSK_IPAR_SIM_SWITCH_OPTIMIZER</td>
<td>Controls the simplex behavior.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>USE_BASIS_EST</td>
<td>Use MOSEK basis estimation in case of an interior solution.</td>
<td>0</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>FIXOPTFILE</td>
<td>Name of option file which is read just before solving the fixed problem.</td>
<td></td>
</tr>
<tr>
<td>MSK_DPAR_MIO_DISABLE_TERM_TIME</td>
<td>Disables termination criteria.</td>
<td>-1.0</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_HEURISTIC_TIME</td>
<td>Maximum amount of time to be used in the heuristic search for a good feasible integer solution.</td>
<td>-1.0</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_MAX_TIME</td>
<td>This parameter limits the maximum time spent by the mixed-integer optimizer.</td>
<td>-1.0</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_MAX_TIME_APRX_OPT</td>
<td>Number of seconds spent by the MIO before the MIO_TOL_REL_RELAX_INT is applied.</td>
<td>60</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_NEAR_TOL_ABS_GAP</td>
<td>Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.</td>
<td>GAMS OptCa</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_NEAR_TOL_REL_GAP</td>
<td>Relaxed relative optimality tolerance employed by the mixed integer optimizer.</td>
<td>GAMS OptCr</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_REL_ADD_CUT_LIMITED</td>
<td>Controls how many cuts the mixed-integer optimizer is allowed to add to the problem.</td>
<td>0.75</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_REL_GAP_CONST</td>
<td>This value is used to compute the relative gap for the solution to an integer optimization problem.</td>
<td>1.0e-10</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_ABS_GAP</td>
<td>Absolute optimality tolerance employed by the mixed-integer optimizer.</td>
<td>0.0</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_ABS_RELAX_INT</td>
<td>Absolute relaxation tolerance of the integer constraints.</td>
<td>1.0e-5</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_FEAS</td>
<td>Feasibility tolerance for mixed integer solver.</td>
<td>1.0e-7</td>
</tr>
<tr>
<td>Environment</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_REL_GAP</td>
<td>Relative optimality tolerance employed by the mixed-integer optimizer.</td>
<td>0.0</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_REL_RELAX_INT</td>
<td>Relative relaxation tolerance of the integer constraints.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>MSK_DPAR_MIO_TOL_X</td>
<td>Absolute solution tolerance used in mixed-integer optimizer.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_BRANCH_DIR</td>
<td>Controls whether the mixed-integer optimizer is branching up or down by default.</td>
<td>MSK_BRANCH_DIR_FREE</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CONSTRUCT_SOL</td>
<td>Controls construction of an initial mixed integer solution from the values of the integer variables</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CONT_SOL</td>
<td>Controls the meaning of the interior-point and basic solutions in mixed integer problems.</td>
<td>MSK_MIO_CONT_SOL_NONE</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CUT_LEVEL_ROOT</td>
<td>Controls the cut level employed by the mixed-integer optimizer at the root node.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_CUT_LEVEL_TREE</td>
<td>Controls the cut level employed by the mixed-integer optimizer at the tree.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_FEASPUMP_LEVEL</td>
<td>Controls the feasibility pump heuristic used to construct a good initial feasible solution.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_HEURISTIC_LEVEL</td>
<td>Controls the heuristic employed by the MIO to locate an initial good integer feasible solution.</td>
<td>-1</td>
</tr>
<tr>
<td>MSK_IPAR_MIO_HOTSTART</td>
<td>Controls whether the integer optimizer is hot-started.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MOSEK IPAR MIO KEEP BASIS</td>
<td>Controls whether the integer presolve keeps bases in memory.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>MOSEK IPAR MIO LOCAL BRANCH NUMBER</td>
<td>Controls the size of the local search space when doing local branching.</td>
<td>-1</td>
</tr>
<tr>
<td>MOSEK IPAR MIO MAX NUM BRANCHES</td>
<td>Maximum number of branches allowed during the branch and bound search.</td>
<td>-1</td>
</tr>
<tr>
<td>MOSEK IPAR MIO MAX NUM RELAXS</td>
<td>Maximum number of relaxations allowed during the branch and bound search.</td>
<td>-1</td>
</tr>
<tr>
<td>MOSEK IPAR MIO MAX NUM SOLUTIONS</td>
<td>Controls how many feasible solutions the mixed-integer optimizer investigates.</td>
<td>-1</td>
</tr>
<tr>
<td>MOSEK IPAR MIO NODE OPTIMIZER</td>
<td>Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.</td>
<td>MSK_OPTIMIZER_FREE</td>
</tr>
<tr>
<td>MOSEK IPAR MIO NODE SELECTION</td>
<td>Controls the node selection strategy employed by the mixed-integer optimizer.</td>
<td>MSK_MIO_NODE_SELECTION_FREE</td>
</tr>
<tr>
<td>MOSEK IPAR MIO PRESOLVE AGGREGATE</td>
<td>Controls whether the presolve used by the MIO tries to aggregate the constraints.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MOSEK IPAR MIO PRESOLVE PROBING</td>
<td>Controls whether the mixed-integer presolve performs probing which can be very time consuming.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MOSEK IPAR MIO PRESOLVE USE</td>
<td>Controls whether presolve is performed by the mixed-integer optimizer.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>MSK_IPAR_MIO_ROOT_OPTIMIZER</strong></td>
<td>Controls which optimizer is employed at the root node in the mixed-integer optimizer.</td>
<td><strong>MSK_OPTIMIZER_FREE</strong></td>
</tr>
<tr>
<td><strong>MSK_IPAR_MIO_STRONG_BRANCH</strong></td>
<td>The value specifies the depth from the root in which strong branching is used.</td>
<td>-1</td>
</tr>
<tr>
<td><strong>MSK_IPAR_MIO_USE_MULTITHREADED_OPTIMIZER</strong></td>
<td>Controls whether the new multithreaded optimizer should be used for Mixed integer problems.</td>
<td><strong>MSK_OFF</strong></td>
</tr>
<tr>
<td><strong>SOLVEFINAL</strong></td>
<td>Switch to solve the problem with fixed discrete variables. Overwrites <strong>MSK_IPAR_MIO_CONT_SOL</strong>.</td>
<td>1</td>
</tr>
</tbody>
</table>

### 4.7 Other Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSK_DPAR_NONCONVEX_TOL_FEAS</strong></td>
<td>Feasibility tolerance used by the nonconvex optimizer.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td><strong>MSK_DPAR_NONCONVEX_TOL_OPT</strong></td>
<td>Optimality tolerance used by the nonconvex optimizer.</td>
<td>1.0e-7</td>
</tr>
<tr>
<td><strong>MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL</strong></td>
<td>Determines when columns are dropped in incomplete cholesky factorization.</td>
<td>1e-15</td>
</tr>
<tr>
<td><strong>MSK_IPAR_ANA_SOL_BASIS</strong></td>
<td>Controls whether the basis matrix is analyzed in solution analyzer.</td>
<td><strong>MSK_ON</strong></td>
</tr>
<tr>
<td><strong>MSK_IPAR_ANA_SOL_PRINT_VIOLATED</strong></td>
<td>Controls whether a list of violated constraints is printed when calling task <strong>ANALYZESOLUTION</strong>.</td>
<td><strong>MSK_OFF</strong></td>
</tr>
<tr>
<td><strong>MSK_IPAR_FEASREPAIR_OPTIMIZE</strong></td>
<td>Controls which type of feasibility analysis is to be performed.</td>
<td><strong>MSK_FEASREPAIR_OPTIMIZE_NONE</strong></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>MSK_IPAR_INFEAS_GENERIC_NAMES</td>
<td>Controls whether generic names are used when an infeasible subproblem is created.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_INFEAS_PREFER_PRIMAL</td>
<td>Controls which certificate is used if primal- and dual-certificate of infeasibility is available.</td>
<td>MSK_ON</td>
</tr>
<tr>
<td>MSK_IPAR_MT_SPINCOUNT</td>
<td>Set the number of iterations to spin before sleeping.</td>
<td>0</td>
</tr>
<tr>
<td>MSK_IPAR_NONCONVEX_MAX_ITERATIONS</td>
<td>Maximum number of iterations that can be used by the nonconvex optimizer.</td>
<td>100000</td>
</tr>
<tr>
<td>MSK_IPAR_OPF_MAX_TERMS_PER_LINE</td>
<td>The maximum number of terms (linear and quadratic) per line when an OPF file is written.</td>
<td>5</td>
</tr>
<tr>
<td>MSK_IPAR_OPF_WRITE_PARAMETERS</td>
<td>Write a parameter section in an OPF file.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_QO_SEPARABLE_REFORMULATION</td>
<td>Determine if Quadratic programing problems should be reformulated to separable form.</td>
<td>MSK_OFF</td>
</tr>
<tr>
<td>MSK_IPAR_SENSITIVITY_OPTIMIZER</td>
<td>Controls which optimizer is used for optimal partition sensitivity analysis.</td>
<td>MSK_OPTIMIZER_FREE_SIMPLEX</td>
</tr>
<tr>
<td>MSK_IPAR_SENSITIVITY_TYPE</td>
<td>Controls which type of sensitivity analysis is to be performed.</td>
<td>MSK_SENSITIVITY_TYPE_BASIS</td>
</tr>
<tr>
<td>MSK_IPAR_TIMING_LEVEL</td>
<td>Controls the amount of timing performed inside MOSEK.</td>
<td>1</td>
</tr>
<tr>
<td>MSK_IPAR_WRITE_DATA_FORMAT</td>
<td>Controls the file format when writing task data to a file.</td>
<td>MSK_DATA_FORMAT_EXTENSION</td>
</tr>
</tbody>
</table>
**5 Detailed Descriptions of MOSEK Options**

**FIXOPTFILE** *(string)*: Name of option file which is read just before solving the fixed problem.

**MSK_DPAR_ANA_SOL_INFEAS_TOL** *(real)*: Report constraint bound violation if it exceeds this tolerance.

  If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

  (default = 1.0e-6)

**MSK_DPAR_BASIS_REL_TOL_S** *(real)*: Maximum relative dual bound violation allowed in an optimal basic solution.

  (default = 1.0e-12)

**MSK_DPAR_BASIS_TOL_S** *(real)*: Maximum absolute dual bound violation in an optimal basic solution.

  (default = 1.0e-6)

**MSK_DPAR_BASIS_TOL_X** *(real)*: Maximum absolute primal bound violation allowed in an optimal basic solution.

  (default = 1.0e-6)

**MSK_DPAR_CHECK_CONVEXITY_REL_TOL** *(real)*: This parameter controls when the full convexity check declares a problem to be non-convex.

  Increasing this tolerance relaxes the criteria for declaring the problem non-convex. A problem is declared non-convex if negative (positive) pivot elements are detected in the cholesky factor of a matrix which is required to be PSD (NSD). This parameter controls how much this non-negativity requirement may be violated. If $d_i$ is the pivot element for column $i$, then the matrix $Q$ is considered to not be PSD if:

  $$d_i <= - |Q_{ii}| \ast \text{MSK_DPAR_CHECK_CONVEXITY_REL_TOL}$$

  (default = 1e-10)

**MSK_DPAR_DATA_TOL_AIJ** *(real)*: Absolute zero tolerance for elements in $A$.
If any value in the constraint matrix is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Range: $[1.0\times10^{-16}, 1.0\times10^{-6}]$

(default = $1.0\times10^{-12}$)

**MSK_DPAR_DATA_TOL_AIJ_HUGE (real)**: An element in $As$ which is larger than this value in absolute size causes an error.

(default = $1.0\times10^{20}$)

**MSK_DPAR_DATA_TOL_AIJ_LARGE (real)**: An element in $As$ which is larger than this value in absolute size causes a warning message.

(default = $1.0\times10^{10}$)

**MSK_DPAR_DATA_TOL_BOUND_INF (real)**: Any bound which in absolute value is greater than this parameter is considered infinite.

(default = $1.0\times10^{16}$)

**MSK_DPAR_DATA_TOL_BOUND_WRN (real)**: If a bound value is larger than this value in absolute size, then a warning message is issued.

(default = $1.0\times10^{8}$)

**MSK_DPAR_DATA_TOL_CJ_LARGE (real)**: An element in $cs$ which is larger than this value in absolute terms causes a warning message.

(default = $1.0\times10^{8}$)

**MSK_DPAR_DATA_TOL_C_HUGE (real)**: An element in $cs$ larger than the value of this parameter in absolute terms generates an error.

c is the objective and coefficients larger than this value are considered to be huge.

(default = $1.0\times10^{16}$)

**MSK_DPAR_DATA_TOL_QIJ (real)**: Absolute zero tolerance for elements in $Q$s matrices.

(default = $1.0\times10^{-16}$)

**MSK_DPAR_DATA_TOL_X (real)**: Zero tolerance for constraints and variables.

That means if the distance between the lower and upper bound is less than this value, then the lower and lower bound is considered identical.

(default = $1.0\times10^{-8}$)

**MSK_DPAR_FEASREPAIR_TOL (real)**: Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

Range: $[1.0\times10^{-16}, 1.0\times10^{16}]$

(default = $1.0\times10^{-10}$)

**MSK_DPAR_INTPNT_CO_TOL_DFEAS (real)**: Dual feasibility tolerance used by the conic interior-point optimizer.

Range: $[0.0, 1.0]$

(default = $1.0\times10^{-8}$)

See also: MSK_DPAR_INTPNT_CO_TOL_NEAR_REL.

**MSK_DPAR_INTPNT_CO_TOL_INFEAS (real)**: Controls when the conic interior-point optimizer declares the model primal or dual infeasible.

A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: $[0.0, 1.0]$

(default = $1.0\times10^{-10}$)

**MSK_DPAR_INTPNT_CO_TOL_MU_RED (real)**: Optimality tolerance for the conic solver.
Relative complementarity gap tolerance feasibility tolerance used by the conic interior-point optimizer.

Range: $[0.0, 1.0]$

(default = $1.0 \times 10^{-8}$)

**MSK_DPAR_INTPNT_CO_TOL_NEAR_REL (real):** Termination tolerance multiplier that is used if no accurate solution can be found.

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

(default = 1000)

**MSK_DPAR_INTPNT_CO_TOL_PFEAS (real):** Primal feasibility tolerance used by the conic interior-point optimizer.

Range: $[0.0, 1.0]$

(default = $1.0 \times 10^{-8}$)

See also: MSK_DPAR_INTPNT_CO_TOL_NEAR_REL.

**MSK_DPAR_INTPNT_CO_TOL_REL_GAP (real):** Relative gap termination tolerance used by the conic interior-point optimizer.

Range: $[0.0, 1.0]$

(default = $1.0 \times 10^{-7}$)

See also: MSK_DPAR_INTPNT_CO_TOL_NEAR_REL.

**MSK_DPAR_INTPNT_NLMerit_BAL (real):** Controls if the complementarity and infeasibility is converging to zero at about equal rates.

Range: $[0.0, 0.99]$

(default = $1.0 \times 10^{-4}$)

**MSK_DPAR_INTPNT_NL_TOL_DFEAS (real):** Dual feasibility tolerance used when a nonlinear model is solved.

Range: $[0.0, 1.0]$

(default = $1.0 \times 10^{-8}$)

**MSK_DPAR_INTPNT_NL_TOL_MU_RED (real):** Relative complementarity gap tolerance.

Range: $[0.0, 1.0]$

(default = $1.0 \times 10^{-12}$)

**MSK_DPAR_INTPNT_NL_TOL_NEAR_REL (real):** Termination tolerance multiplier that is used if no accurate solution can be found.

If MOSEK nonlinear interior-point optimizer cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

(default = 1000.0)

**MSK_DPAR_INTPNT_NL_TOL_PFEAS (real):** Primal feasibility tolerance used when a nonlinear model is solved.

Range: $[0.0, 1.0]$

(default = $1.0 \times 10^{-8}$)

**MSK_DPAR_INTPNT_NL_TOL_REL_GAP (real):** Relative gap termination tolerance for nonlinear problems.

(default = $1.0 \times 10^{-6}$)

**MSK_DPAR_INTPNT_NL_TOL_REL_STEP (real):** Relative step size to the boundary for general nonlinear optimization problems.

Range: $[1.0 \times 10^{-4}, 0.9999999]$
MSK_DPAR_INTPNT_TOL_DFEAS \( (\text{real}) \): Dual feasibility tolerance used for linear and quadratic optimization problems.

Range: \([0.0, 1.0]\)

(default = 1.0e-8)

MSK_DPAR_INTPNT_TOL_DSAFE \( (\text{real}) \): Controls the initial dual starting point used by the interior-point optimizer.

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

(default = 1.0)

MSK_DPAR_INTPNT_TOL_INFEAS \( (\text{real}) \): Controls when the optimizer declares the model primal or dual infeasible.

A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: \([0.0, 1.0]\)

(default = 1.0e-10)

MSK_DPAR_INTPNT_TOL_MU_RED \( (\text{real}) \): Relative complementarity gap tolerance.

Range: \([0.0, 1.0]\)

(default = 1.0e-16)

MSK_DPAR_INTPNT_TOL_PATH \( (\text{real}) \): Controls how close the interior-point optimizer follows the central path.

A large value of this parameter means the central is followed very closely. On numerical unstable problems it might worthwhile to increase this parameter.

Range: \([0.0, 0.9999]\)

(default = 1.0e-8)

MSK_DPAR_INTPNT_TOL_PFEAS \( (\text{real}) \): Primal feasibility tolerance used for linear and quadratic optimization problems.

Range: \([0.0, 1.0]\)

(default = 1.0e-8)

MSK_DPAR_INTPNT_TOL_PSAFE \( (\text{real}) \): Controls the initial primal starting point used by the interior-point optimizer.

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

(default = 1.0)

MSK_DPAR_INTPNT_TOL_REL_GAP \( (\text{real}) \): Relative gap termination tolerance.

(default = 1.0e-8)

MSK_DPAR_INTPNT_TOL_REL_STEP \( (\text{real}) \): Relative step size to the boundary for linear and quadratic optimization problems.

Range: \([1.0e-4, 0.999999]\)

(default = 0.99999)

MSK_DPAR_INTPNT_TOL_STEP_SIZE \( (\text{real}) \): Step size tolerance value.

If the step size falls below the value of this parameter, then the interior-point optimizer assumes it is stalled. It it does not not make any progress.

Range: \([0.0, 1.0]\)

(default = 1.0e-6)

MSK_DPAR_LOWER_OBJ_CUT \( (\text{real}) \): Lower objective limit.

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval \((\text{MSK_DPAR_LOWER_OBJ_CUT}, \text{MSK_DPAR_UPPER_OBJ_CUT})\), then MOSEK is terminated.
See also: MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH.

**MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH** *(real)*: Lower objective limit threshold.

If the lower objective cut (MSK_DPAR_LOWER_OBJ_CUT) is less than MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH, then the lower objective cut MSK_DPAR_LOWER_OBJ_CUT is treated as infinity.

*(default = \(-0.5\times 10^{-30}\))*

**MSK_DPAR_MIO_DISABLE_TERM_TIME** *(real)*: Disables termination criteria.

The termination criteria governed by

- MSK_IPAR_MIO_MAX_NUM_RELAXS
- MSK_IPAR_MIO_MAX_NUM_BRANCHES
- MSK_DPAR_MIO_NEAR_TOL_ABS_GAP
- MSK_DPAR_MIO_NEAR_TOL_REL_GAP

are disabled the first \(n\) seconds. This parameter specifies the number \(n\). A negative value is identical to infinity i.e. the termination criteria are never checked.

*(default = \(-1.0\))*

See also: MSK_DPAR_MIO_NEAR_TOL_ABS_GAP, MSK_DPAR_MIO_NEAR_TOL_REL_GAP, MSK_IPAR_MIO_MAX_NUM_BRANCHES.

**MSK_DPAR_MIO_HEURISTIC_TIME** *(real)*: Maximum amount of time to be used in the heuristic search for a good feasible integer solution.

A negative values implies that the optimizer decides the amount of time to be spend in the heuristic.

*(default = \(-1.0\))*

**MSK_DPAR_MIO_MAX_TIME** *(real)*: This parameter limits the maximum time spent by the mixed-integer optimizer.

A negative number means infinity.

*(default = \(-1.0\))*

**MSK_DPAR_MIO_MAX_TIME_APRX_OPT** *(real)*: Number of seconds spent by the MIO before the MIO TOL_REL_RELAX_INT is applied.

*(default = 60)*

**MSK_DPAR_MIO_NEAR_TOL_ABS_GAP** *(real)*: Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.

*Synonym: optca*

This termination criteria is delayed.

*(default = GAMS OptCa)*

See also: MSK_DPAR_MIO_DISABLE_TERM_TIME.

**MSK_DPAR_MIO_NEAR_TOL_REL_GAP** *(real)*: Relaxed relative optimality tolerance employed by the mixed integer optimizer.

*Synonym: optcr*

The mixed integer optimizer is terminated when this tolerance is satisfied. This termination criteria is delayed. See MSK_DPAR_MIO_DISABLE_TERM_TIME for details.

*(default = GAMS OptCr)*

See also: MSK_DPAR_MIO_DISABLE_TERM_TIME.

**MSK_DPAR_MIO_REL_ADD_CUTLIMITED** *(real)*: Controls how many cuts the mixed-integer optimizer is allowed to add to the problem.

Let \(s\) be the number of constraints, then the mixed-integer optimizer is allowed to add
MSK_DPAR_MIO_REL_ADD_CUT_LIMITED $m$ cuts.
Range: [0.0, 2.0]
(default = 0.75)

**MSK_DPAR_MIO_REL_GAP_CONST** *(real)*: This value is used to compute the relative gap for the solution to an integer optimization problem.
(default = 1.0e-10)

**MSK_DPAR_MIO_TOL_ABS_GAP** *(real)*: Absolute optimality tolerance employed by the mixed-integer optimizer.
(default = 0.0)

**MSK_DPAR_MIO_TOL_ABS_RELAX_INT** *(real)*: Absolute relaxation tolerance of the integer constraints.
That means if the fractional part of a discrete variable is less than the tolerance, the integer restrictions assumed to be satisfied.
(default = 1.0e-5)

**MSK_DPAR_MIO_TOL_FEAS** *(real)*: Feasibility tolerance for mixed integer solver.
Any solution with maximum infeasibility below this value will be considered feasible.
(default = 1.0e-7)

**MSK_DPAR_MIO_TOL_REL_GAP** *(real)*: Relative optimality tolerance employed by the mixed-integer optimizer.
(default = 0.0)

**MSK_DPAR_MIO_TOL_REL_RELAX_INT** *(real)*: Relative relaxation tolerance of the integer constraints.
That means if the fractional part of a discrete variable is less than the tolerance times the level of that variable, the integer restrictions assumed to be satisfied.
(default = 1.0e-6)

**MSK_DPAR_MIO_TOL_X** *(real)*: Absolute solution tolerance used in mixed-integer optimizer.
(default = 1.0e-6)

**MSK_DPAR_NONCONVEX_TOL_FEAS** *(real)*: Feasibility tolerance used by the nonconvex optimizer.
(default = 1.0e-6)

**MSK_DPAR_NONCONVEX_TOL_OPT** *(real)*: Optimality tolerance used by the nonconvex optimizer.
(default = 1.0e-7)

**MSK_DPAR_OPTIMIZER_MAX_TIME** *(real)*: Maximum amount of time the optimizer is allowed to spent on the optimization.
Synonym: reslim
A negative number means infinity.
(default = GAMS ResLim)

**MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP** *(real)*: Absolute tolerance employed by the linear dependency checker.
(default = 1.0e-6)

**MSK_DPAR_PRESOLVE_TOL_AIJ** *(real)*: Absolute zero tolerance employed for $a_{ij}$ in the presolve.
(default = 1.0e-12)

**MSK_DPAR_PRESOLVE_TOL_REL_LINDEP** *(real)*: Relative tolerance employed by the linear dependency checker.
(default = 1.0e-10)

**MSK_DPAR_PRESOLVE_TOL_S** *(real)*: Absolute zero tolerance employed for $s_j$ in the presolve.
(default = 1.0e-8)

**MSK_DPAR_PRESOLVE_TOL_X** *(real)*: Absolute zero tolerance employed for $x_j$ in the presolve.
**MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL** *(real)*: Determines when columns are dropped in incomplete cholesky factorization.

This parameter is applied when doing reformulation of quadratic problems.

(default = 1.0e-15)

**MSK_DPAR_SIMPLEX_ABS_TOL_PIV** *(real)*: Absolute pivot tolerance employed by the simplex optimizers.

(default = 1.0e-7)

**MSK_DPAR_SIM_LU_TOL_REL_PIV** *(real)*: Relative pivot tolerance for LU factorization in simplex and basis identification.

This tolerance is employed when computing the LU factorization of the basis in the simplex optimizers and in the basis identification procedure. A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Range: [1.0e-6, 0.999999]

(default = 0.01)

**MSK_DPAR_UPPER_OBJ_CUT** *(real)*: Upper objective limit.

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval (MSK_DPAR_LOWER_OBJ_CUT, MSK_DPAR_UPPER_OBJ_CUT), then MOSEK is terminated.

(default = 1.0e30)

See also: MSK_DPAR_UPPER_OBJ_CUT_FINE_FTRH.

**MSK_DPAR_UPPER_OBJ_CUT_FINE_TRH** *(real)*: Upper objective limit threshold.

If the upper objective cut (MSK_DPAR_UPPER_OBJ_CUT) is greater than MSK_DPAR_UPPER_OBJ_CUT_FINE_TRH, then the upper objective cut MSK_DPAR_UPPER_OBJ_CUT is treated as infinity.

(default = 0.5e30)

**MSK_IPAR_ANA_SOL_BASIS** *(string)*: Controls whether the basis matrix is analyzed in solution analyzer.

(default = MSK_ON)

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_ANA_SOL_PRINT_VIOLATED** *(string)*: Controls whether a list of violated constraints is printed when calling task ANALYZESOLUTION.

All constraints violated by more than the value set by the parameter MSK_DPAR_ANA_SOL_INFEAS_TOL will be printed.

(default = MSK_OFF)

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_AUTO_SORT_A_BEFORE_OPT** *(string)*: Controls whether the elements in each column of $A$ are sorted before an optimization is performed.

This is not required but makes the optimization more deterministic.

(default = MSK_OFF)

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_BI_CLEAN_OPTIMIZER** *(string)*: Controls which optimizer is used in the clean-up phase.

(default = MSK_OPTIMIZER_FREE)
MSK_OPTIMIZER_CONCURRENT The optimizer for nonconvex nonlinear problems.
MSK_OPTIMIZER_CONIC The optimizer for problems having conic constraints.
MSK_OPTIMIZER_DUAL_SIMPLEX The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE The optimizer is chosen automatically.
MSK_OPTIMIZER_FREE_SIMPLEX One of the simplex optimizers is used.
MSK_OPTIMIZER_INTPNT The interior-point optimizer is used.
MSK_OPTIMIZER_MIXED_INT The mixed-integer optimizer.
MSK_OPTIMIZER_MIXED_INT_CONIC The mixed-integer optimizer for conic and linear problems.
MSK_OPTIMIZER_NETWORK_PRIMAL_SIMPLEX The network primal simplex optimizer is used. It is only applicable to network problems.
MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.
MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX The primal dual simplex optimizer is used.
MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

**MSK_IPAR_BI_IGNORE_MAX_ITER** *(string)*: Controls if basis identification is performed under certain conditions.

If the parameter MSK_IPAR_INTPNT_BASIS has the value MSK_BI_NO_ERROR and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value MSK_ON.

*(default = MSK_OFF)*

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_BI_IGNORE_NUM_ERROR** *(string)*: Turns on basis identification if interior-point optimizer is terminated due to a numerical problem.

If the parameter MSK_IPAR_INTPNT_BASIS has the value MSK_BI_NO_ERROR and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value MSK_ON.

*(default = MSK_OFF)*

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_BI_MAX_ITERATIONS** *(integer)*: Maximum number of iterations after basis identification.

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

*(default = 1000000)*

**MSK_IPAR_CHECK_CONVEXITY** *(string)*: Specify the level of convexity check on quadratic problems

*(default = MSK_CHECK_CONVEXITY_FULL)*

- **MSK_CHECK_CONVEXITY_FULL** Perform a full convexity check.
- **MSK_CHECK_CONVEXITY_NONE** No convexity check.
- **MSK_CHECK_CONVEXITY_SIMPLE** Perform simple and fast convexity check.

**MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS** *(integer)*: The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.

*(default = 2)*

**MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX** *(integer)*: Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.

*(default = 2)*
MSK_IPAR_INFEAS_REPORT_AUTO (string): Controls infeasibility report production. Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

(default = MSK_OFF)
  MSK_OFF Switch the option off.
  MSK_ON Switch the option on.

MSK_IPAR_INFEAS_REPORT_LEVEL (integer): Controls the amount of information presented in an infeasibility report. Higher values imply more information.

(default = 1)

MSK_IPAR_INTPNT_BASIS (string): Controls whether the interior-point optimizer also computes an optimal basis.

(default = MSK_BI_ALWAYS)
  MSK_BI_ALWAYS Always perform basis identification. Basis identification is always performed even if the interior-point optimizer terminates abnormally.
  MSK_BI_IF_FEASIBLE Only perform Basis identification if problem status is feasible. Basis identifi-
cation is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.
MOSEK 387

MSK_BI_NEVER Never do basis identification.
MSK_BI_NO_ERROR Basis identification is performed if the interior-point optimizer terminates without an error.

See also: MSK_IPAR_BI_IGNORE_MAX_ITER, MSK_IPAR_BI_IGNORE_NUM_ERROR.

**MSK_IPAR_INTPNT_DIFF_STEP** *(string)*: Controls whether different step sizes are allowed in the primal and dual space.

(default = MSK_OFF)
- MSK_OFF Switch the option off.
- MSK_ON Switch the option on.

**MSK_IPAR_INTPNT_FACTOR_DEBUG_LVL** *(integer)*: Controls factorization debug level.

(default = 0)

**MSK_IPAR_INTPNT_FACTOR_METHOD** *(integer)*: Controls the method used to factor the Newton equation system.

(default = 0)

**MSK_IPAR_INTPNT_MAX_ITERATIONS** *(integer)*: Controls the maximum number of iterations allowed in the interior-point optimizer.

(default = 400)

**MSK_IPAR_INTPNT_MAX_NUM_COR** *(integer)*: Controls the maximum number of correctors allowed by the multiple corrector procedure.

A negative value means that MOSEK is making the choice.

(default = -1)

**MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS** *(integer)*: Maximum number of steps to be used by the iterative refinement of the search direction.

A negative value implies that the optimizer chooses the maximum number of iterative refinement steps.

(default = -1)

**MSK_IPAR_INTPNT_OFF_COL_TRH** *(integer)*: Controls how many offending columns are detected in the Jacobian of the constraint matrix.

1 means aggressive detection, higher values mean less aggressive detection. 0 means no detection.

(default = 40)

**MSK_IPAR_INTPNT_ORDER_METHOD** *(string)*: Controls the ordering strategy.

This refers to the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

(default = MSK_ORDER_METHOD_FREE)
- MSK_ORDER_METHOD_APPMINLOC Approximate minimum local fill-in ordering is employed.
- MSK_ORDER_METHOD_EXPERIMENTAL This option should not be used.
- MSK_ORDER_METHOD_FORCE_GRAPHPAR Always use the graph partitioning based ordering. Use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.
- MSK_ORDER_METHOD_FREE The ordering method is chosen automatically.
- MSK_ORDER_METHOD_NONE No ordering is used.
- MSK_ORDER_METHOD_TRY_GRAPHPAR Always try the the graph partitioning based ordering.

**MSK_IPAR_INTPNT_REGULARIZATION_USE** *(string)*: Controls whether regularization is allowed.

(default = MSK_ON)
- MSK_OFF Switch the option off.
- MSK_ON Switch the option on.
**MSK_IPAR_INTPNT_SCALING** *(string)*: Controls how the problem is scaled before the interior-point optimizer is used.

(default = **MSK_SCALING_FREE**)

- **MSK_SCALING_AGGRESSIVE** A very aggressive scaling is performed.
- **MSK_SCALING_FREE** The optimizer chooses the scaling heuristic.
- **MSK_SCALING_MODERATE** A conservative scaling is performed.
- **MSK_SCALING_NONE** No scaling is performed.

**MSK_IPAR_INTPNT_SOLVE_FORM** *(string)*: Controls whether the primal or the dual problem is solved.

(default = **MSK_SOLVE_FREE**)

- **MSK_SOLVE_DUAL** The optimizer should solve the dual problem.
- **MSK_SOLVE_FREE** The optimizer is free to solve either the primal or the dual problem.
- **MSK_SOLVE_PRIMAL** The optimizer should solve the primal problem.

**MSK_IPAR_INTPNT_STARTING_POINT** *(string)*: Starting point used by the interior-point optimizer.

(default = **MSK_STARTING_POINT_FREE**)

- **MSK_STARTING_POINT_CONSTANT** The optimizer constructs a starting point by assigning a constant value to all variables. The constant value is assigned to all primal and dual variables. This starting point is normally robust.
- **MSK_STARTING_POINT_FREE** The starting point is chosen automatically.
- **MSK_STARTING_POINT_GUESS** The optimizer guesses a starting point.
- **MSK_STARTING_POINT_SATISFY_BOUNDS** The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should be exercised when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

**MSK_IPAR_LOG** *(integer)*: Controls the amount of log information.

The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

(default = 10)

**MSK_IPAR_LOG_BI** *(integer)*: Controls the amount of output printed by the basis identification procedure.

A higher level implies that more information is logged.

(default = 4)

**MSK_IPAR_LOG_BI_FREQ** *(integer)*: Controls logging frequency.

In detail this option controls how frequent the optimizer outputs information about the basis identification and how frequent the user-defined call-back function is called.

(default = 2500)

**MSK_IPAR_LOG_CHECK_CONVEXITY** *(integer)*: Controls logging in convexity check on quadratic problems.

Set to a positive value to turn logging on. If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

(default = 0)

**MSK_IPAR_LOG_CONCURRENT** *(integer)*: Controls amount of output printed by the concurrent optimizer.

(default = 1)

**MSK_IPAR_LOG_FACTOR** *(integer)*: If turned on, then the factor log lines are added to the log.

(default = 1)
**MSK_IPAR_LOG_FEAS_REPAIR** *(integer)*: Controls the amount of output printed when performing feasibility repair.

    (default = 1)

**MSK_IPAR_LOG_HEAD** *(integer)*: If turned on, then a header line is added to the log.

    (default = 1)

**MSK_IPAR_LOG_INFEAS_ANA** *(integer)*: Controls amount of output printed by the infeasibility analyzer procedures.

    A higher level implies that more information is logged.

    (default = 1)

**MSK_IPAR_LOG_INTPNT** *(integer)*: Controls amount of output printed printed by the interior-point optimizer.

    A higher level implies that more information is logged.

    (default = 4)

**MSK_IPAR_LOG_MIO** *(integer)*: Controls the log level for the mixed-integer optimizer.

    A higher level implies that more information is logged.

    (default = 4)

**MSK_IPAR_LOG_MIO_FREQ** *(integer)*: Controls how frequent the mixed-integer optimizer prints the log line.

    It will print a line every time MSK_INTPAR_LOG_MIO_FREQ relaxations have been solved.

    (default = 1000)

**MSK_IPAR_LOG_NONCONVEX** *(integer)*: Controls amount of output printed by the nonconvex optimizer.

    (default = 1)

**MSK_IPAR_LOG_OPTIMIZER** *(integer)*: Controls the amount of general optimizer information that is logged.

    (default = 1)

**MSK_IPAR_LOG_ORDER** *(integer)*: If turned on, then factor lines are added to the log.

    (default = 1)

**MSK_IPAR_LOG_PARAM** *(integer)*: Controls the amount of information printed out about parameter changes.

    (default = 0)

**MSK_IPAR_LOG_PRESOLVE** *(integer)*: Controls amount of output printed by the presolve procedure.

    A higher level implies that more information is logged.

    (default = 1)

**MSK_IPAR_LOG_RESPONSE** *(integer)*: Controls amount of output printed when response codes are reported.

    A higher level implies that more information is logged.

    (default = 0)

**MSK_IPAR_LOG_SENSITIVITY** *(integer)*: Controls the amount of logging during the sensitivity analysis.

    • 0: No logging information is produced.
    • 1: Timing information is printed.
    • 2: Sensitivity results are printed.

    (default = 1)

**MSK_IPAR_LOG_SENSITIVITY_OPT** *(integer)*: Controls the amount of logging from the optimizers employed during the sensitivity analysis.

    0 means no logging information is produced.

    (default = 0)

**MSK_IPAR_LOG_SIM** *(integer)*: Controls amount of output printed by the simplex optimizer.
A higher level implies that more information is logged.
(default = 4)

**MSK_IPAR_LOG_SIM_FREQ (integer):** Controls simplex logging frequency.

In detail this option controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined call-back function is called.

(default = 1000)

**MSK_IPAR_LOG_STORAGE (integer):** When turned on, MOSEK prints messages regarding the storage usage and allocation.

(default = 0)

**MSK_IPAR_MAX_NUM_WARNINGS (integer):** Warning level. A higher value results in more warnings.

(default = 10)

**MSK_IPAR_MIO_BRANCH_DIR (string):** Controls whether the mixed-integer optimizer is branching up or down by default.

(default = MSK_BRANCH_DIR_FREE)
- **MSK_BRANCH_DIR_DOWN** The mixed-integer optimizer always chooses the down branch first.
- **MSK_BRANCH_DIR_FREE** The mixed-integer optimizer decides which branch to choose.
- **MSK_BRANCH_DIR_UP** The mixed-integer optimizer always chooses the up branch first.

**MSK_IPAR_MIO_CONSTRUCT_SOL (string):** Controls construction of an initial mixed integer solution from the values of the integer variables

Synonym: mipstart
If set to MSK_ON and all integer variables have been given a value for which a feasible mixed integer solution exists, then MOSEK generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

(default = MSK_OFF)
- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_MIO_CONT_SOL (string):** Controls the meaning of the interior-point and basic solutions in mixed integer problems.

(default = MSK_MIO_CONT_SOL_NONE)
- **MSK_MIO_CONT_SOL_ITG** The solution is to the problem with all discrete variables fixed.
- **MSK_MIO_CONT_SOL_NONE** The fixed problem is skipped. The primal integer solution is reported.
- **MSK_MIO_CONT_SOL_ROOT** The LP solution to the root node problem is reported.
- **MSK_MIO_CONT_SOL_ITG_REL** Reports the solution to the fixed problem for feasible problems otherwise the root solution.

**MSK_IPAR_MIO_CUT_LEVEL_ROOT (integer):** Controls the cut level employed by the mixed-integer optimizer at the root node.

A negative value means a default value determined by the mixed integer optimizer is used. By adding the appropriate values from the following table the employed cut types can be controlled.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUB cover</td>
<td>+2</td>
</tr>
<tr>
<td>Flow cover</td>
<td>+4</td>
</tr>
<tr>
<td>Lifting</td>
<td>+8</td>
</tr>
<tr>
<td>Plant location</td>
<td>+16</td>
</tr>
<tr>
<td>Disaggregation</td>
<td>+32</td>
</tr>
<tr>
<td>Knapsack cover</td>
<td>+64</td>
</tr>
<tr>
<td>Lattice</td>
<td>+128</td>
</tr>
</tbody>
</table>
MSK_IPAR_MIO_CUT_LEVEL_TREE (integer): Controls the cut level employed by the mixed-integer optimizer at the tree.

See MSK_IPAR_MIO_CUT_LEVEL_ROOT for an explanation of the parameter values.

(default = -1)

MSK_IPAR_MIO_FEASPUMP_LEVEL (integer): Controls the feasibility pump heuristic used to construct a good initial feasible solution.

A value of 0 implies that the feasibility pump heuristic is not used. A value of -1 implies that the mixed integer optimizer decides how the feasibility pump heuristic is used. A larger value than 1 implies that the feasibility pump is employed more aggressively. Normally a value beyond 3 is not worthwhile.

Range: [minint, 3]

(default = -1)

MSK_IPAR_MIO_HEURISTIC_LEVEL (integer): Controls the heuristic employed by the MIO to locate an initial good integer feasible solution.

A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

(default = -1)

MSK_IPAR_MIO_HOTSTART (string): Controls whether the integer optimizer is hot-started.

(default = MSK_OFF)

  MSK_OFF Switch the option off.

  MSK_ON Switch the option on.

MSK_IPAR_MIO_KEEP_BASIS (string): Controls whether the integer presolve keeps bases in memory.

This speeds on the solution process at cost of bigger memory consumption.

(default = MSK_OFF)

  MSK_OFF Switch the option off.

  MSK_ON Switch the option on.

MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER (integer): Controls the size of the local search space when doing local branching.

(default = -1)

MSK_IPAR_MIO_MAX_NUM_BRANCHES (integer): Maximum number of branches allowed during the branch and bound search.

A negative value means infinite.

(default = -1)

See also: MSK_DPAR_MIO_DISABLE_TERM_TIME.

MSK_IPAR_MIO_MAX_NUM_RELAXS (integer): Maximum number of relaxations allowed during the branch and bound search.

Synonym: nodlim

A negative value means infinite.

(default = -1)
See also: MSK::DPAR::MIO::DISABLE::TERM::TIME.

**MSK::IPAR::MIO::MAX::NUM::SOLUTIONS** (*integer*): Controls how many feasible solutions the mixed-integer optimizer investigates.

The mixed integer optimizer can be terminated after a certain number of different feasible solutions have been located. If this parameter has the value $n$ and $n$ is strictly positive, then the mixed integer optimizer will be terminated when $n$ feasible solutions have been located.

(default = -1)

See also: MSK::DPAR::MIO::DISABLE::TERM::TIME.

**MSK::IPAR::MIO::NODE::OPTIMIZER** (*string*): Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

(default = MSK::OPTIMIZER::FREE)

- `MSK::OPTIMIZER::CONCURRENT` The optimizer for nonconvex nonlinear problems.
- `MSK::OPTIMIZER::CONIC` The optimizer for problems having conic constraints.
- `MSK::OPTIMIZER::DUAL::SIMPLEX` The dual simplex optimizer is used.
- `MSK::OPTIMIZER::FREE` The optimizer is chosen automatically.
- `MSK::OPTIMIZER::FREE::SIMPLEX` One of the simplex optimizers is used.
- `MSK::OPTIMIZER::INTPNT` The interior-point optimizer is used.
- `MSK::OPTIMIZER::MIXED::INT` The mixed-integer optimizer.
- `MSK::OPTIMIZER::MIXED::INT::CONIC` The mixed-integer optimizer for conic and linear problems.
- `MSK::OPTIMIZER::NETWORK::PRIMAL::SIMPLEX` The network primal simplex optimizer is used. It is only applicable to pure network problems.
- `MSK::OPTIMIZER::NONCONVEX` The optimizer for nonconvex nonlinear problems.
- `MSK::OPTIMIZER::PRIMAL::DUAL::SIMPLEX` The primal dual simplex optimizer is used.
- `MSK::OPTIMIZER::PRIMAL::SIMPLEX` The primal simplex optimizer is used.

**MSK::IPAR::MIO::NODE::SELECTION** (*string*): Controls the node selection strategy employed by the mixed-integer optimizer.

(default = MSK::MIO::NODE::SELECTION::FREE)

- `MSK::MIO::NODE::SELECTION::BEST` The optimizer employs a best bound node selection strategy.
- `MSK::MIO::NODE::SELECTION::FIRST` The optimizer employs a depth first node selection strategy.
- `MSK::MIO::NODE::SELECTION::FREE` The optimizer decides the node selection strategy.
- `MSK::MIO::NODE::SELECTION::HYBRID` The optimizer employs a hybrid strategy.
- `MSK::MIO::NODE::SELECTION::PSEUDO` The optimizer employs selects the node based on a pseudo cost estimate.
- `MSK::MIO::NODE::SELECTION::WORST` The optimizer employs a worst bound node selection strategy.

**MSK::IPAR::MIO::PRESOLVE::AGGREGATE** (*string*): Controls whether the presolve used by the MIO tries to aggregate the constraints.

(default = MSK::ON)

- `MSK::OFF` Switch the option off.
- `MSK::ON` Switch the option on.

**MSK::IPAR::MIO::PRESOLVE::PROBING** (*string*): Controls whether the mixed-integer presolve performs probing which can be very time consuming.

(default = MSK::ON)
MSK_OFF Switch the option off.
MSK_ON Switch the option on.

**MSK_IPAR_MIO_PRESOLVE_USE (string):** Controls whether presolve is performed by the mixed-integer optimizer.
   (default = MSK_ON)
   MSK_OFF Switch the option off.
   MSK_ON Switch the option on.

**MSK_IPAR_MIO_ROOT_OPTIMIZER (string):** Controls which optimizer is employed at the root node in the mixed-integer optimizer.
   (default = MSK_OPTIMIZER_FREE)
   MSK_OPTIMIZER_CONCURRENT The optimizer for nonconvex nonlinear problems.
   MSK_OPTIMIZER_CONIC The optimizer for problems having conic constraints.
   MSK_OPTIMIZER_DUAL_SIMPLEX The dual simplex optimizer is used.
   MSK_OPTIMIZER_FREE The optimizer is chosen automatically.
   MSK_OPTIMIZER_FREE_SIMPLEX One of the simplex optimizers is used.
   MSK_OPTIMIZER_INTPNT The interior-point optimizer is used.
   MSK_OPTIMIZER_MIXED_INT The mixed-integer optimizer.
   MSK_OPTIMIZER_MIXED_INT_CONIC The mixed-integer optimizer for conic and linear problems.
   MSK_OPTIMIZER_NETWORK_PRIMAL_SIMPLEX The network primal simplex optimizer is used. It is only applicable to pure network problems.
   MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.
   MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX The primal dual simplex optimizer is used.
   MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

**MSK_IPAR_MIO_STRONG_BRANCH (integer):** The value specifies the depth from the root in which strong branching is used.
   A negative value means the optimizer chooses a default value automatically.
   (default = -1)

**MSK_IPAR_MIO_USE_MULTITHREADED_OPTIMIZER (string):** Controls whether the new multithreaded optimizer should be used for Mixed integer problems.
   (default = MSK_OFF)
   MSK_OFF Switch the option off.
   MSK_ON Switch the option on.

**MSK_IPAR_MT_SPINCOUNT (integer):** Set the number of iterations to spin before sleeping.
   Range: [0, 1000000000]
   (default = 0)

**MSK_IPAR_NONCONVEX_MAX_ITERATIONS (integer):** Maximum number of iterations that can be used by the nonconvex optimizer.
   (default = 100000)

**MSK_IPAR_NUM_THREADS (integer):** Controls the number of threads employed by the optimizer.
   If set to 0 the number of threads used will be equal to the number of cores detected on the machine.
   (default = 0)

**MSK_IPAR_OPF_MAX_TERMS_PER_LINE (integer):** The maximum number of terms (linear and quadratic) per line
when an OPF file is written.

(default = 5)

**MSK_IPAR_OPF_WRITE_PARAMETERS (string):** Write a parameter section in an OPF file.

(default = MSK_OFF)

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_OPTIMIZER (string):** The parameter controls which optimizer is used to optimize the task.

(default = MSK_OPTIMIZER_FREE)

- **MSK_OPTIMIZER_CONCURRENT** The optimizer for nonconvex nonlinear problems.
- **MSK_OPTIMIZER_CONIC** The optimizer for problems having conic constraints.
- **MSK_OPTIMIZER_DUAL_SIMPLEX** The dual simplex optimizer is used.
- **MSK_OPTIMIZER_FREE** The optimizer is chosen automatically.
- **MSK_OPTIMIZER_FREE_SIMPLEX** One of the simplex optimizers is used.
- **MSK_OPTIMIZER_INTPNT** The interior-point optimizer is used.
- **MSK_OPTIMIZER_MIXED_INT** The mixed-integer optimizer.
- **MSK_OPTIMIZER_MIXED_INT_CONIC** The mixed-integer optimizer for conic and linear problems.
- **MSK_OPTIMIZER_NETWORK_PRIMAL_SIMPLEX** The network primal simplex optimizer is used. It is only applicable to pure network problems.
- **MSK_OPTIMIZER_NONCONVEX** The optimizer for nonconvex nonlinear problems.
- **MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX** The primal dual simplex optimizer is used.
- **MSK_OPTIMIZER_PRIMAL_SIMPLEX** The primal simplex optimizer is used.

**MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES (integer):** Control the maximum number of times the eliminator is tried.

(default = -1)

**MSK_IPAR_PRESOLVE_ELIMINATOR_USE (string):** Controls whether free or implied free variables are eliminated from the problem.

(default = MSK_ON)

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_PRESOLVE_ELIM_FILL (integer):** Controls the maximum amount of fill-in in the elimination phase of the presolve.

This parameter times the number of variables plus the number of constraints denotes the amount of fill in.

(default = 1)

**MSK_IPAR_PRESOLVE_LEVEL (integer):** Currently not used.

(default = -1)

**MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH (integer):** Controls the linear dependency check which is potentially computationally expensive.

(default = 100)

**MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH (integer):** Controls the linear dependency check which is potentially computationally expensive.

(default = 100)
**MSK_IPAR_PRESOLVE_LINDEP_USE (string):** Controls whether the linear constraints are checked for linear dependencies.

(default = MSK_ON)

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS (integer):** Controls the maximum number reductions performed by the presolve.

The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

(default = -1)

**MSK_IPAR_PRESOLVE_USE (string):** Controls whether the presolve is applied to a problem before it is optimized.

(default = MSK_PRESOLVE_MODE_FREE)

- **MSK_PRESOLVE_MODE_FREE** It is decided automatically whether to presolve before the problem is optimized.
- **MSK_PRESOLVE_MODE_OFF** The problem is not presolved before it is optimized.
- **MSK_PRESOLVE_MODE_ON** The problem is presolved before it is optimized.

**MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER (string):** Controls which optimizer that is used to find the optimal repair.

(default = MSK_OPTIMIZER_FREE)

- **MSK_OPTIMIZER_CONCURRENT** The optimizer for nonconvex nonlinear problems.
- **MSK_OPTIMIZER_CONIC** The optimizer for problems having conic constraints.
- **MSK_OPTIMIZER_DUAL_SIMPLEX** The dual simplex optimizer is used.
- **MSK_OPTIMIZER_FREE** The optimizer is chosen automatically.
- **MSK_OPTIMIZER_FREE_SIMPLEX** One of the simplex optimizers is used.
- **MSK_OPTIMIZER_INTPNT** The interior-point optimizer is used.
- **MSK_OPTIMIZER_MIXED_INT** The mixed-integer optimizer.
- **MSK_OPTIMIZER_MIXED_INT_CONIC** The mixed-integer optimizer for conic and linear problems.
- **MSK_OPTIMIZER_NETWORK_PRIMAL_SIMPLEX** The network primal simplex optimizer is used. It is only applicable to pure network problems.
- **MSK_OPTIMIZER_NONCONVEX** The optimizer for nonconvex nonlinear problems.
- **MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX** The primal dual simplex optimizer is used.
- **MSK_OPTIMIZER_PRIMAL_SIMPLEX** The primal simplex optimizer is used.

**MSK_IPAR_QO_SEPARABLE_REFORMULATION (string):** Determine if Quadratic programming problems should be reformulated to separable form.

(default = MSK_OFF)

- **MSK_OFF** Switch the option off.
- **MSK_ON** Switch the option on.

**MSK_IPAR_SENSITIVITY_OPTIMIZER (string):** Controls which optimizer is used for optimal partition sensitivity analysis.

(default = MSK_OPTIMIZER_FREE_SIMPLEX)

- **MSK_OPTIMIZER_CONCURRENT** The optimizer for nonconvex nonlinear problems.
- **MSK_OPTIMIZER_CONIC** The optimizer for problems having conic constraints.
- **MSK_OPTIMIZER_DUAL_SIMPLEX** The dual simplex optimizer is used.
The optimizer is chosen automatically.

The optimizer for nonconvex nonlinear problems.

The primal simplex optimizer is used.

The primal dual simplex optimizer is used.

The network primal simplex optimizer is used. It is only applicable to pure network problems.

The mixed-integer optimizer.

The mixed-integer optimizer for conic and linear problems.

The simple sensitivity analysis is to be performed.

Basis sensitivity analysis is performed.

Optimal partition sensitivity analysis is performed.

Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Controls whether crashing is performed in the dual simplex optimizer. In general if a basis consists of more than (100*MSK_IPAR_SIM_DUAL_CRASH) percent fixed variables, then a crash will be performed.

Controls how aggressively degeneration is handled.

The simplex optimizer should use an aggressive degeneration strategy.

The simplex optimizer chooses the degeneration strategy.

The simplex optimizer should use a minimum degeneration strategy.

The simplex optimizer should use a moderate degeneration strategy.

The simplex optimizer should use no degeneration strategy.

Controls whether restricted selection is used. The dual simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Controls the dual simplex strategy.

In detail this option controls the choice of the incoming variable, known as the selection strategy, in the dual
simplex optimizer.

(default = MSK_SIM_SELECTION_FREE)

- MSK_SIM_SELECTION_ASE: The optimizer uses approximate steepest-edge pricing.
- MSK_SIM_SELECTION_DEVEX: The optimizer uses devex steepest-edge pricing. If it is not available an approximate steep-edge selection is chosen.
- MSK_SIM_SELECTION_FREE: The optimizer chooses the pricing strategy.
- MSK_SIM_SELECTION_FULL: The optimizer uses full pricing.
- MSK_SIM_SELECTION_PARTIAL: The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.
- MSK_SIM_SELECTION_SE: The optimizer uses steepest-edge selection. If it is not available an approximate steep-edge selection is chosen.

**MSK_IPAR_SIM_EXPLOIT_DUPVEC (string):** Controls if the simplex optimizers are allowed to exploit duplicated columns.

(default = MSK_SIM_EXPLOIT_DUPVEC_OFF)

- MSK_SIM_EXPLOIT_DUPVEC_FREE: The simplex optimizer can choose freely.
- MSK_SIM_EXPLOIT_DUPVEC_OFF: Disallow the simplex optimizer to exploit duplicated columns.
- MSK_SIM_EXPLOIT_DUPVEC_ON: Allow the simplex optimizer to exploit duplicated columns.

**MSK_IPAR_SIM_HOTSTART (string):** Controls the type of hot-start that the simplex optimizer performs.

(default = MSK_SIM_HOTSTART_FREE)

- MSK_SIM_HOTSTART_FREE: The simplex optimizer chooses the hot-start type.
- MSK_SIM_HOTSTART_NONE: The simplex optimizer performs a coldstart.
- MSK_SIM_HOTSTART_STATUS_KEYS: Only the status keys of the constraints and variables are used to choose the type of hot-start.

**MSK_IPAR_SIM_HOTSTART_LU (string):** Determines if the simplex optimizer should exploit the initial factorization.

(default = MSK_ON)

- MSK_OFF: Switch the option off.
- MSK_ON: Switch the option on.

**MSK_IPAR_SIM_MAX_ITERATIONS (integer):** Maximum number of iterations that can be used by a simplex optimizer.

Synonym: iterlim

(default = GAMS IterLim)

**MSK_IPAR_SIM_MAX_NUM_SETBACKS (integer):** Controls how many set-backs are allowed within a simplex optimizer.

A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

(default = 250)

**MSK_IPAR_SIM_NON_SINGULAR (string):** Controls if the simplex optimizer ensures a non-singular basis, if possible.

(default = MSK_ON)

- MSK_OFF: Switch the option off.
- MSK_ON: Switch the option on.

**MSK_IPAR_SIM_PRIMAL_CRASH (integer):** Controls whether crashing is performed in the primal simplex optimizer.

In general if a basis consists of more than \((100 + MSK_IPAR_SIM_PRIMAL_CRASH)\) percent fixed variables, then a crash will be performed.
(default = GAMS BRatio)

**MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION** (integer): Controls how aggressively restricted selection is used.

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Range: \([0, 100]\)

(default = 50)

**MSK_IPAR_SIM_PRIMAL_SELECTION** (string): Controls the primal simplex strategy.

In detail this option controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

(default = MSK_SIM_SELECTION_FREE)

- **MSK_SIM_SELECTION_ASE**: The optimizer uses approximate steepest-edge pricing.
- **MSK_SIM_SELECTION_DEVEX**: The optimizer uses devex steepest-edge pricing. If it is not available an approximate steep-edge selection is chosen.
- **MSK_SIM_SELECTION_FREE**: The optimizer chooses the pricing strategy.
- **MSK_SIM_SELECTION_FULL**: The optimizer uses full pricing.
- **MSK_SIM_SELECTION_PARTIAL**: The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.
- **MSK_SIM_SELECTION_SE**: The optimizer uses steepest-edge selection. If it is not available an approximate steep-edge selection is chosen.

**MSK_IPAR_SIM_REFACTOR_FREQ** (integer): Controls how frequent the basis is refactored.

The value 0 means that the optimizer determines the best point of refactorization. It is strongly recommended NOT to change this parameter.

(default = 0)

**MSK_IPAR_SIM_REFORMULATION** (string): Controls if the simplex optimizers are allowed to reformulate the problem.

(default = MSK_SIM_REFORMULATION_OFF)

- **MSK_SIM_REFORMULATION_AGGRESSIVE**: The simplex optimizer should use an aggressive reformulation strategy.
- **MSK_SIM_REFORMULATION_FREE**: The simplex optimizer can choose freely.
- **MSK_SIM_REFORMULATION_OFF**: Disallow the simplex optimizer to reformulate the problem.
- **MSK_SIM_REFORMULATION_ON**: Allow the simplex optimizer to reformulate the problem.

**MSK_IPAR_SIM_SAVE_LU** (string): Controls storage of LU factorization.

In detail this option controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

(default = MSK_OFF)

- **MSK_OFF**: Switch the option off.
- **MSK_ON**: Switch the option on.

**MSK_IPAR_SIM_SCALING** (string): Controls how much effort is used in scaling the problem before a simplex optimizer is used.

(default = MSK_SCALING_FREE)
MSK_SCALING_AGGRESSIVE A very aggressive scaling is performed.
MSK_SCALING_FREE The optimizer chooses the scaling heuristic.
MSK_SCALING_MODERATE A conservative scaling is performed.
MSK_SCALING_NONE No scaling is performed.

**MSK_IPAR_SIM_SCALING_METHOD** *(string)*: Controls how the problem is scaled before a simplex optimizer is used.
(default = MSK_SCALING_METHOD_POW2)
- MSK_SCALING_METHOD_FREE The optimizer chooses the scaling heuristic.
- MSK_SCALING_METHOD_POW2 Scales only with power of 2 leaving the mantissa untouched.

**MSK_IPAR_SIM_SOLVE_FORM** *(string)*: Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.
(default = MSK_SOLVE_FREE)
- MSK_SOLVE_DUAL The optimizer should solve the dual problem.
- MSK_SOLVE_FREE The optimizer is free to solve either the primal or the dual problem.
- MSK_SOLVE_PRIMAL The optimizer should solve the primal problem.

**MSK_IPAR_SIM_STABILITY_PRIORITY** *(integer)*: Controls how high priority the numerical stability should be given.
Range: [0, 100]
(default = 50)

**MSK_IPAR_SIM_SWITCH_OPTIMIZER** *(string)*: Controls the simplex behavior.
The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).
(default = MSK_OFF)
- MSK_OFF Switch the option off.
- MSK_ON Switch the option on.

**MSK_IPAR_TIMING_LEVEL** *(integer)*: Controls the amount of timing performed inside MOSEK.
(default = 1)

**MSK_IPAR_WARNING_LEVEL** *(integer)*: Warning level. A higher value implies more warnings.
(default = 1)

**MSK_IPAR_WRITE_DATA_FORMAT** *(string)*: Controls the file format when writing task data to a file.
(default = MSK_DATA_FORMAT_EXTENSION)

**MSK_IPAR_WRITE_GENERIC_NAMES** *(string)*: Controls whether the generic names or user-defined names are used in the data file.
(default = MSK_OFF)
- MSK_OFF Switch the option off.
- MSK_ON Switch the option on.

**MSK_IPAR_WRITE_GENERIC_NAMES_IO** *(integer)*: Index origin used in generic names.
(default = 1)

**MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS** *(string)*: Controls ignoreation of incompatible conic items.
If the output format is not compatible with conic quadratic problems this parameter controls if the writer ignores the conic parts or produces an error.
MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS (string): Controls if the writer ignores incompatible problem items when writing files.

(default = MSK_OFF)

  MSK_OFF Switch the option off.
  MSK_ON Switch the option on.

MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS (string): Controls if the writer ignores general non-linear terms or produces an error.

(default = MSK_OFF)

  MSK_OFF Switch the option off.
  MSK_ON Switch the option on.

MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS (string): Controls ignorance of incompatible psd items.

If the output format is not compatible with semidefinite problems this parameter controls if the writer ignores the psd parts or produces an error.

(default = MSK_OFF)

  MSK_OFF Switch the option off.
  MSK_ON Switch the option on.

MSK_IPAR_WRITE_LP_LINE_WIDTH (integer): Maximum width of line in an LP file written by MOSEK.

(default = 80)

MSK_IPAR_WRITE_LP_QUOTED_NAMES (string): If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.

(default = MSK_ON)

  MSK_OFF Switch the option off.
  MSK_ON Switch the option on.

MSK_IPAR_WRITE_LP STRICT_FORMAT (string): Controls whether LP output files satisfy the LP format strictly.

(default = MSK_OFF)

  MSK_OFF Switch the option off.
  MSK_ON Switch the option on.

MSK_IPAR_WRITE_LP_TERMS_PER_LINE (integer): Maximum number of terms on a single line in an LP file written by MOSEK. 0 means unlimited.

(default = 10)

MSK_IPAR_WRITE_MPS_INT (string): Controls writing of marker records to the MPS file to indicate if variables are integer restricted.

(default = MSK_ON)

  MSK_OFF Switch the option off.
  MSK_ON Switch the option on.

MSK_IPAR_WRITE_PRECISION (integer): Controls the precision with which double numbers are printed in the MPS data file.

  In general it is not worthwhile to use a value higher than 15.

  (default = 8)
MSK_IPAR_WRITE_XML_MODE (string): Controls output mode for XML file.
   In detail this option controls if linear coefficients should be written by row or column when writing in the XML
   file format.
   (default = MSK_WRITE_XML_MODE_ROW)

MSK_SPAR_DATA_FILE_NAME (string): Problem data is written to this file. File extension determines format.
   Synonym: writeprob

MSK_SPAR_PARAM_READ_FILE_NAME (string): Modifications to the parameter database is read from this file.

MSK_SPAR_PARAM_WRITE_FILE_NAME (string): The parameter database is written to this file.

READFILE (string): Read secondary option file.

SDPCHECKVARS (integer): Switch to disable checking that for every entry of a PSD matrix variable also a corresponding
   GAMS variable is present.
   (default = 1)

SDPSOLUFILE (string): Name of GDX file to write primal solution of all PSD matrix variables and dual solution for PSD
   constraints.

SOLVEFINAL (integer): Switch to solve the problem with fixed discrete variables. Overwrites MSK_IPAR_MIO_CONT_SOL.
   (default = 1)
   0 Do not solve the fixed problem
   1 Solve the fixed problem and return duals

USE_BASIS_EST (integer): Use MOSEK basis estimation in case of an interior solution.
   (default = 0)

6 The MOSEK Log File

The MOSEK log output gives much useful information about the current solver progress and individual phases.

6.1 Log Using the Interior Point Optimizer

The following is a MOSEK log output from running the transportation model transport.gms from the GAMS Model
Library:

Optimizer started.
Interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries : 0 time : 0.00
Eliminator - elim's : 0
Lin. dep. - tries : 1 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
Optimizer - threads : 1
Optimizer - solved problem : the primal
Optimizer - Constraints : 5
Optimizer - Cones : 0
Optimizer - Scalar variables : 11 conic : 0
Optimizer - Semi-definite variables: 0 scalarized : 0
Factor - setup time : 0.00 dense det. time : 0.00
The first part gives information about the presolve (if used). The main log follows:

<table>
<thead>
<tr>
<th>ITE</th>
<th>PFEAS</th>
<th>DFEAS</th>
<th>GFEAS</th>
<th>PRSTATUS</th>
<th>POBJ</th>
<th>DOBJ</th>
<th>MU</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6.0e+002</td>
<td>1.0e+002</td>
<td>1.0e+002</td>
<td>1.0530000000e+000</td>
<td>0.0000000000e+000</td>
<td>1.0e+002</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5.9e+002</td>
<td>1.3e+002</td>
<td>2.6e+002</td>
<td>0.0000000000e+000</td>
<td>2.650041612e+002</td>
<td>2.4e+002</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.6e+001</td>
<td>1.0e+001</td>
<td>2.0e+001</td>
<td>-9.59e-001</td>
<td>3.650704301e+001</td>
<td>2.594816940e+002</td>
<td>1.9e+001</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>3.9e-001</td>
<td>8.7e-002</td>
<td>1.7e-001</td>
<td>-4.39e-001</td>
<td>1.604589379e+002</td>
<td>2.276036163e+002</td>
<td>1.6e-001</td>
<td>0.01</td>
</tr>
<tr>
<td>4</td>
<td>2.7e-002</td>
<td>6.0e-003</td>
<td>1.2e-002</td>
<td>9.62e-001</td>
<td>1.627664502e+002</td>
<td>1.676438787e+002</td>
<td>1.1e-002</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>2.2e-003</td>
<td>4.9e-004</td>
<td>9.7e-004</td>
<td>1.04e+000</td>
<td>1.585004810e+002</td>
<td>1.591499235e+002</td>
<td>8.9e-004</td>
<td>0.01</td>
</tr>
<tr>
<td>6</td>
<td>3.1e-004</td>
<td>6.9e-005</td>
<td>1.4e-004</td>
<td>1.01e+000</td>
<td>1.546312243e+002</td>
<td>1.547272945e+002</td>
<td>1.2e-004</td>
<td>0.01</td>
</tr>
<tr>
<td>7</td>
<td>2.9e-005</td>
<td>6.5e-006</td>
<td>1.3e-005</td>
<td>1.01e+000</td>
<td>1.536906429e+002</td>
<td>1.536996285e+002</td>
<td>1.2e-005</td>
<td>0.01</td>
</tr>
<tr>
<td>8</td>
<td>7.6e-008</td>
<td>1.7e-008</td>
<td>3.4e-008</td>
<td>1.00e+000</td>
<td>1.536751995e+002</td>
<td>1.536752387e+002</td>
<td>3.1e-008</td>
<td>0.01</td>
</tr>
<tr>
<td>9</td>
<td>7.6e-012</td>
<td>1.7e-012</td>
<td>3.4e-012</td>
<td>1.00e+000</td>
<td>1.536750000e+002</td>
<td>1.536750000e+002</td>
<td>3.1e-012</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Basis identification started.

Basis identification started.

Iter    Time
1        0.00

Basis identification terminated. Time: 0.00

Dual basis identification phase started.

Iter    Time
0        0.00

Basis identification terminated. Time: 0.00

Interior-point optimizer terminated. Time: 0.01.

Optimizer terminated. Time: 0.03

Interior-point solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL

Primal. obj: 1.5367500002e+002 Viol. con: 3e-010 var: 0e+000
Dual. obj: 1.5367500002e+002 Viol. con: 0e+000 var: 6e-011

Basic solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL

Primal. obj: 1.5367500000e+002 Viol. con: 0e+000 var: 0e+000
Dual. obj: 1.5367500000e+002 Viol. con: 0e+000 var: 5e-011

Return code - 0 [MSK_RES_OK]: No error occurred.

The last section gives details about the model and solver status, primal and dual feasibilities, as well as solver resource times. Furthermore, the log gives information about the basis identification phase. Some of this information is listed in the GAMS solve summary in the model listing (.LST) file as well.

The fields in the main MOSEK log output are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITE</td>
<td>The number of the current iteration.</td>
</tr>
<tr>
<td>PFEAS</td>
<td>Primal feasibility.</td>
</tr>
<tr>
<td>DFEAS</td>
<td>Dual feasibility.</td>
</tr>
</tbody>
</table>
Below is a log output running the model transport.gms from the GAMS model library using the MOSEK simplex optimizer.

Reading parameter(s) from "c:\tmp\mosek.opt"
>> MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_DUAL_SIMPLEX
Finished reading from "c:\tmp\mosek.opt"
Optimizer started.
Simplex optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries : 0 time : 0.00
Eliminator - elim's : 0
Lin. dep. - tries : 1 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
Dual simplex optimizer started.
Dual simplex optimizer setup started.
Dual simplex optimizer setup terminated.
Optimizer - solved problem : the primal
Optimizer - Constraints : 5
Optimizer - Scalar variables : 6 conic : 0
Optimizer - hotstart : no
ITER DEGITER(%) PFEAS DFEAS POBJ DOBJ TIME TOTTIME
0 0.00 NA 0.00e+000 NA 0.00000000000e+000 0.00 0.02
4 20.00 NA 0.00e+000 NA 1.5367501014e+002 0.00 0.02
Dual simplex optimizer terminated.
Simplex optimizer terminated. Time: 0.00.
Optimizer terminated. Time: 0.02

Basic solution summary
  Problem status : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal. obj: 1.5367500000e+002 Viol. con: 0e+000 var: 0e+000
  Dual. obj: 1.5367500000e+002 Viol. con: 0e+000 var: 0e+000

Return code - 0 [MSK_RES_OK]: No error occurred.

The fields in the main MOSEK log output are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFEAS</td>
<td>The numbers in this column should converge monotonically toward to zero but may stall at low level due to rounding errors.</td>
</tr>
<tr>
<td>PRSTATUS</td>
<td>This number converge to 1 if the problem has an optimal solution whereas it converge to -1 if that is not the case.</td>
</tr>
<tr>
<td>POBJ</td>
<td>Current objective function value of primal problem.</td>
</tr>
<tr>
<td>DOBJ</td>
<td>Current objective function value of dual problem.</td>
</tr>
<tr>
<td>MU</td>
<td>Relative complementary gap.</td>
</tr>
<tr>
<td>TIME</td>
<td>Current elapsed resource time in seconds.</td>
</tr>
</tbody>
</table>
6.3 Log Using the Mixed Integer Optimizer

Below is a log output running the model cube.gms from the GAMS model library using the MOSEK mixed-integer optimizer.

Optimizer started.
Mixed integer optimizer started.

```
BRANCHES RELAXS ACT_NDS BEST_INT_OBJ BEST_RELAX_OBJ REL_GAP(%) TIME
0     0     0     18.0000000000e+000 NA       NA       0.0       0.0
0     1     0     18.0000000000e+000 0.0000000000e+000 100.00 1.0
0     1     0     7.0000000000e+000 0.0000000000e+000 100.00 1.0
0     10    0     6.0000000000e+000 0.0000000000e+000 100.00 1.0
0     15    0     4.0000000000e+000 0.0000000000e+000 100.00 1.0
355   1003   42    4.0000000000e+000 0.0000000000e+000 100.00 2.1
392   1081   45    4.0000000000e+000 5.0000000000e-001 87.50 2.2
```

A near optimal solution satisfying the absolute gap tolerance of 3.90e+000 has been located.

The fields in the main MOSEK log output are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRANCHES</td>
<td>Current number of branches in tree.</td>
</tr>
<tr>
<td>RELAXS</td>
<td>Current number of nodes in branch and bound tree.</td>
</tr>
<tr>
<td>ACT_NDS</td>
<td>Current number of active nodes.</td>
</tr>
<tr>
<td>BEST_INT_OBJ</td>
<td>Current best integer solution</td>
</tr>
<tr>
<td>BEST_RELAX_OBJ</td>
<td>Current best relaxed solution.</td>
</tr>
<tr>
<td>REL_GAP(%)</td>
<td>Relative gap between current BEST_INT_OBJ. and BEST_RELAX_OBJ.</td>
</tr>
<tr>
<td>TIME</td>
<td>Current elapsed resource time in seconds.</td>
</tr>
</tbody>
</table>

The log then gives information about solving the model with discrete variables fixed in order to determine marginals and some summary of the branch and cut algorithm. We also get information about crossover to determine a basic solution, and finally MOSEK provides information about using the Simplex Method to determine an optimal basic solution.

Interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.

```
Eliminator - tries : 0 time : 0.00
Eliminator - elim's : 31
Lin. dep. - tries : 1 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
```

Interior-point optimizer terminated. Time: 0.00.
Objective of best integer solution : 4.000000000000e+000
Construct solution objective : Not employed
User objective cut value : Not employed
Number of cuts generated : 24
  Number of Gomory cuts : 9
  Number of contra cuts : 2
Number of branches : 392
Number of relaxations solved : 1081
Number of interior point iterations: 0
Number of simplex iterations : 17710
Time spend presolving the root : 0.02
Time spend in the heuristic : 0.56
Time spend in the sub optimizers : 1.59
  Time spend optimizing the root : 0.00
Mixed integer optimizer terminated. Time: 2.23
Optimizer terminated. Time: 2.23

Interior-point solution summary
  Problem status : PRIMAL_FEASIBLE
  Solution status : PRIMAL_FEASIBLE
  Primal. obj: 4.0000000000e+000 Viol. con: 0e+000 var: 0e+000 itg: 0e+000
  Dual. obj: -6.0000000000e+001 Viol. con: 0e+000 var: 0e+000

Basic solution summary
  Problem status : PRIMAL_FEASIBLE
  Solution status : PRIMAL_FEASIBLE
  Primal. obj: 4.0000000000e+000 Viol. con: 0e+000 var: 0e+000 itg: 0e+000
  Dual. obj: -6.0000000000e+001 Viol. con: 0e+000 var: 0e+000

Integer solution solution summary
  Problem status : PRIMAL_FEASIBLE
  Solution status : NEAR_INTEGER_OPTIMAL
  Primal. obj: 4.0000000000e+000 Viol. con: 0e+000 var: 0e+000 itg: 0e+000

Return code - 10004 [MSK_RES_TRM_MIO_NEAR_ABS_GAP]: The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

MIP Solution: 4.000000 (0 iterations, 0 nodes)
Best possible: 0.500000
Absolute gap: 3.500000
Relative gap: 3.500000

6.4 Log Using the Conic Mixed Integer Optimizer

Below is a log output running the model cube.gms from the GAMS model library using the MOSEK conic mixed-integer optimizer. The columns of the iteration log are identical to the regular mixed-integer optimizer.

Reading parameter(s) from "c:\tmp\mosek.opt"
>> MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_MIXED_INT_CONIC
Finished reading from "c:\tmp\mosek.opt"
Optimizer started.
Mixed integer optimizer started.
Optimizer - threads : 1
<table>
<thead>
<tr>
<th>BRANCHES</th>
<th>RELAXS</th>
<th>ACT_NDS</th>
<th>BEST_INT_OBJ</th>
<th>BEST_RELAX_OBJ</th>
<th>REL_GAP(%)</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>NA</td>
<td>0.0000000000e+000</td>
<td>NA</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>NA</td>
<td>0.0000000000e+000</td>
<td>NA</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>NA</td>
<td>0.0000000000e+000</td>
<td>NA</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1.2000000000e+001</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>7.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>7.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>7.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>8</td>
<td>7.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>16</td>
<td>7.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>31</td>
<td>33</td>
<td>28</td>
<td>6.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>59</td>
<td>61</td>
<td>48</td>
<td>6.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>89</td>
<td>91</td>
<td>74</td>
<td>6.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>121</td>
<td>123</td>
<td>90</td>
<td>6.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.1</td>
</tr>
<tr>
<td>151</td>
<td>153</td>
<td>112</td>
<td>5.0000000000e+000</td>
<td>0.0000000000e+000</td>
<td>100.00</td>
<td>0.2</td>
</tr>
<tr>
<td>183</td>
<td>184</td>
<td>134</td>
<td>5.0000000000e+000</td>
<td>3.3333333333e-001</td>
<td>93.33</td>
<td>0.2</td>
</tr>
<tr>
<td>215</td>
<td>216</td>
<td>160</td>
<td>5.0000000000e+000</td>
<td>3.3333333333e-001</td>
<td>93.33</td>
<td>0.2</td>
</tr>
<tr>
<td>247</td>
<td>248</td>
<td>184</td>
<td>5.0000000000e+000</td>
<td>5.0000000000e-001</td>
<td>90.00</td>
<td>0.2</td>
</tr>
<tr>
<td>280</td>
<td>280</td>
<td>209</td>
<td>5.0000000000e+000</td>
<td>5.0000000000e-001</td>
<td>90.00</td>
<td>0.2</td>
</tr>
<tr>
<td>313</td>
<td>312</td>
<td>232</td>
<td>5.0000000000e+000</td>
<td>5.0000000000e-001</td>
<td>90.00</td>
<td>0.2</td>
</tr>
<tr>
<td>346</td>
<td>345</td>
<td>251</td>
<td>5.0000000000e+000</td>
<td>5.8823529412e-001</td>
<td>88.24</td>
<td>0.3</td>
</tr>
<tr>
<td>378</td>
<td>377</td>
<td>279</td>
<td>5.0000000000e+000</td>
<td>7.1428571429e-001</td>
<td>85.71</td>
<td>0.3</td>
</tr>
<tr>
<td>410</td>
<td>409</td>
<td>299</td>
<td>5.0000000000e+000</td>
<td>1.0000000000e+000</td>
<td>80.00</td>
<td>0.3</td>
</tr>
<tr>
<td>443</td>
<td>441</td>
<td>318</td>
<td>5.0000000000e+000</td>
<td>1.0000000000e+000</td>
<td>80.00</td>
<td>0.3</td>
</tr>
<tr>
<td>477</td>
<td>475</td>
<td>340</td>
<td>5.0000000000e+000</td>
<td>1.0000000000e+000</td>
<td>80.00</td>
<td>0.3</td>
</tr>
<tr>
<td>510</td>
<td>506</td>
<td>365</td>
<td>4.0000000000e+000</td>
<td>1.0000000000e+000</td>
<td>75.00</td>
<td>0.3</td>
</tr>
</tbody>
</table>

A near optimal solution satisfying the absolute gap tolerance of 3.90e+000 has been located.

Objective of best integer solution : 4.000000000000e+000
Construct solution objective : Not employed
User objective cut value : Not employed
Number of cuts generated : 3
Number of branches : 510
Number of relaxations solved : 506
Number of interior point iterations: 5
Number of simplex iterations : 5581
Time spend presolving the root : 0.00
Time spend in the heuristic : 0.00
Time spend in the sub optimizers : 0.00
Time spend optimizing the root : 0.03
Mixed integer optimizer terminated. Time: 0.34

The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied. (*10004*)
Optimizer terminated. Time: 0.34

Integer solution solution summary
Problem status : PRIMAL_FEASIBLE
Solution status : NEAR_INTEGER_OPTIMAL
Primal. obj: 4.0000000000e+000 Viol. con: 0e+000 var: 0e+000 itg: 0e+000

Return code - 10004 [MSK_RES_TRM_MIO_NEAR_ABS_GAP]: The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.
The log then gives information about solving the model with discrete variables fixed in order to determine marginals. For this section, the fixed problem is solved as a regular LP with warm start information. So the log looks identical to the MOSEK simplex optimizer for linear programs:

Solving fixed problem...

MOSEK 24.1.3 r41646 Released Jul 26, 2013 VS8 x86/MS Windows

Optimizer started.
Simplex optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries : 0 time : 0.00
Eliminator - elim's : 31
Lin. dep. - tries : 1 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
Simplex optimizer terminated. Time: 0.00.

Basic solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal. obj: 4.0000000000e+000 Viol. con: 0e+000 var: 0e+000
Dual. obj: 4.0000000000e+000 Viol. con: 0e+000 var: 0e+000

Return code - 0 [MSK_RES_OK]: No error occurred.

The log ends with the regular summary for MIP problems printing objective values, bounds and gaps:

MIP Solution: 4.000000 (0 iterations, 0 nodes)
Final Solve: 4.000000 (0 iterations)

Best possible: 4.000000
Absolute gap: 0.000000
Relative gap: 0.000000

7 Semidefinite Programming with GAMS/MOSEK (experimental)

With Version 7, MOSEK introduced a semidefinite programming (SDP) solver into their portfolio. The following introduction is taken from the MOSEK manual.

Semidefinite programming is a generalization of quadratic conic programming, allowing the use of matrix variables belonging to the convex cone of positive semidefinite matrices

$$\mathcal{S}_r^+ = \{X \in \mathcal{S}_r : z^T X z \geq 0, \forall z \in \mathbb{R}^r \},$$

where $\mathcal{S}_r$ is the set of $r \times r$ real-valued symmetric matrices. MOSEK can solve semidefinite optimization problems of the
minimize \[ \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle C_j, X_j \rangle + c^f \]

subject to \[ l^c_i \leq \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle A_{ij}, X_j \rangle \leq u^c_i, \quad i = 0, \ldots, m - 1, \]

\[ l^x_j \leq x_j \quad x \in \mathcal{C}, X_j \in \mathcal{S}^r_j^+, \quad j = 0, \ldots, n - 1, \]

\[ l^x_j \leq x_j \quad j = 0, \ldots, p - 1 \]

where the problem has $p$ symmetric positive semidefinite (PSD) variables $X_j \in \mathcal{S}^r_j^+$ of dimension $r_j$ with symmetric coefficient matrices $C_j \in \mathcal{S}_r^+$ and $A_{ij} \in \mathcal{S}_r^+$. We use standard notation for the matrix inner product, i.e., for $A, B \in \mathbb{R}^{m \times n}$ we have

\[ \langle A, B \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{ij} B_{ij}. \]

An example for a mixed semidefinite and conic quadratic programming problem with a 3-dimensional PSD matrix variable is the following:

\[
\begin{align*}
\text{minimize} & \quad \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \bar{X} \right\rangle + x_0 \\
\text{subject to} & \quad \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_0 = 1 \\
& \quad \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_1 + x_2 = \frac{1}{2} \\
& \quad x_0 \geq \sqrt{x_1^2 + x_2^2} \\
& \quad \bar{X} \succeq 0
\end{align*}
\]

7.1 Example

The GAMS/MOSEK interface offers an experimental interface to MOSEK’s SDP solver. It allows to state SDP’s of the form (8) in GAMS language. For example, problem (9) can be written as shown in the following code. We see that matrix $\bar{X}$ is defined via the 2-dimensional GAMS variable $\bar{X}$. Additionally, the keyword PSDMATRIX at the beginning of the descriptive text (!) of the variable is used to indicate that the variables belonging to symbol $\bar{X}$ are to be interpreted as a matrix variable with PSD constraint.

The following is the GAMS formulation for problem (9). This model is available in the GAMS Test library under the name sdp01.
The Model shown above can be solved with MOSEK via the statements

1 option qcp = mosek;
2 Solve m minimizing z using QCP;

As GAMS has no native support for conic programming, the modeltype is specified as QCP.

The solve statement produces the following log output:

MOSEK version 7.0.0.75 (Build date: 2013-7-1 19:28:43)
Copyright (C) MOSEK ApS, Fruebjergvej 3, Box 16
DK-2100 Copenhagen, Denmark
http://www.mosek.com

Recognizing SDP variables for symbols barX (1)
Recognized 1 quadratic equation as conic constraint.
Optimizer started.
Conic interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries : 0 time : 0.00
Eliminator - elim's : 0
Lin. dep. - tries : 1 time : 0.00
Lin. dep. - number : 0
Presolve terminated. Time: 0.00
Optimizer - threads : 1
Optimizer - solved problem : the primal
Optimizer - Constraints : 2
Optimizer - Cones : 1
Optimizer - Scalar variables : 3 conic : 3
Optimizer - Semi-definite variables: 1 scalarized : 6
Factor - setup time : 0.00 dense det. time : 0.00
Factor - ML order time : 0.00 GP order time : 0.00
Factor - nonzeros before factor : 3 after factor : 3
Factor - dense dim. : 0 flops : 2.88e+02
ITE   PFEAS   DFEAS   GFEAS   PRSTATUS   POBJ     DOBJ     MU   TIME
 0  3.0e+00  1.0e+00  8.0e+00  0.00e+00  7.000000000e+00  0.000000000e+00  1.0e+00  0.00
 1  4.3e-01  1.4e-01  1.2e+00  1.67e-01  1.601231478e+00  3.103213067e-01  1.4e-01  0.00
 2  6.6e-02  2.2e-02  1.8e-01  1.21e-01  8.534948745e-01  6.725260078e-01  2.2e-02  0.00
 3  4.9e-03  1.6e-03  1.3e-02  1.02e+00  7.158760069e-01  7.026959645e-01  1.6e-03  0.00
 4  2.8e-04  9.4e-05  7.5e-04  1.00e+00  7.063055286e-01  7.055481573e-01  9.4e-05  0.00
Interior-point optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Interior-point solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL

Primal. obj: 7.0571052965e-01 Viol. con: 2e-08 var: 0e+00 barvar: 0e+00 cones: 0e+00
Dual. obj: 7.0571048621e-01 Viol. con: 0e+00 var: 1e-16 barvar: 0e+00 cones: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.

Finally, the optimal value for the matrix $\bar{X}$ and the dual values associated with bound constraints on entries of the matrix $\bar{X}$ can be displayed in GAMS as usual: display $X.l$, $X.m$;

7.2 Usage

The general syntax for defining a symmetric matrix or an (indexed) set of symmetric matrices with PSD constraints in GAMS is

Variable $X(a,b,c,\ldots,i1,\ldots,i1,i1,\ldots,i1)$ "PSDMATRIX_k <explanatory text>";

The number $k \in \mathbb{N}_{>0}$ specifies the number of indices that define the row/column dimension of the matrix. Specifying only the term PSDMATRIX is equivalent to PSDMATRIX_1. For a given $k$, the $2k$-last indices are used to index the rows and columns of the matrix. The dimensions $a$, $b$, $c$, ... are optional and can be used to index a set of matrices. In the GAMS/MOSEK output, a number in parenthesis is used to indicate the number of PSD matrices that have been found for one symbol. For example, the code

```
1 Set i / a,b /;
2 Set j / s01 * s42 /;
3 Variables z, X(j,i,i) PSDMATRIX;
```

generates the output

Recognizing SDP variables for symbols $X$ (42)

if all matrices $X(j,\ldots)$ also occur in a model instantiation.
Note

The current syntax for declaring PSD matrix variables via the explanatory text a variable is a temporary solution that will hopefully be replaced by a GAMS language feature in the future.

Variables that were tagged as belonging to a PSD matrix can only occur in linear constraints. Within these constraints, the coefficient matrix for a PSD matrix need to specified in symmetric form. That is, if the parameter matrix \( \text{barAobj} \) in the example above is equivalently specified as:

1 Table barAobj(i,i)
2 0 1 2
3 0 2.0 0.0 0.0
4 1 2.0 2.0 0.0
5 2 0.0 2.0 2.0
6 ;

the GAMS/MOSEK interface will quit with the error message:

SDP coefficient matrix not symmetric: defObj_z: 2*barX(1,0) != 0*barX(0,1)

Bounds on entries in a PSD matrix variable can be specified as usual with .lo and .up attributes. These bound constraints are translated into linear constraints by the interface. If different bounds are given to symmetric entries of a PSD matrix variable (\( X_{i,j} \) vs. \( X_{j,i} \)), the stronger bounds are used, which is equivalent to adding constraints for each of the matrix entries. For nonpositive lower bounds on diagonal entries, no extra constraints are added, as they are implied by the PSD constraint.

### 7.3 GAMS variables vs. PSD matrix entries

As GAMS is not aware that a indexed variable will be interpreted as PSD matrix variable, it may not generate variables for all matrix entries when instantiating a model. This is especially critical if the modeler specified bounds on matrix entries that do not appear in any of the model equations, as these bounds would not be visible to the GAMS/MOSEK interface.

As an example, consider the SDP relaxation:

\[
\max \{-\langle W, \bar{X} \rangle : X_{i,i} = 1 \ \forall i, X \succeq 0\}
\]

corresponding to the Goemans-Williamson Randomized Approximation Algorithm for MaxCut:

1 Parameter W(i,i) edge weights;
2 Variable X(i,j) PSDMATRIX
3 sdpobjvar objective var;
4 Equation sdpobj objective function;
5 sdpobj.. sum((i,j), -W(i,j)*(X(i,j) + X(j,i))) =e= sdpobjvar;
6 X.fx(i,i) = 1.0;
7 model maxcutsdp / all /;

As \( W(i,i)=0 \), GAMS will not create any variables for \( X(i,i) \) when instantiating the model in a solve statement. As a consequence, the constraints \( X(i,i)=1 \) will not be visible to GAMS/MOSEK. However, as MOSEK will compute values for the full matrix \( X \), it will also compute values for the diagonal entries of \( X \).

Further, the GAMS/MOSEK interface can pass primal solution values only for those entries of a PSD matrix variable that have a corresponding GAMS variable. This may make a solution matrix appear not to be PSD in GAMS, because not all entries have been passed back.

To be aware of such problems, the GAMS/MOSEK interface checks that it has a GAMS variable available for every entry of a PSD matrix variable. If not, it will report an error like:

ERROR: Have 1600 GAMS variables for entries of 400 x 400 PSD Variable X(,), expected 160000 many.

This check can be disabled by setting the option SDPCHECKVARS to 0. Note, that the check is not able to alarm the user in situations where no GAMS variables were created for all entries of a row and corresponding column.

A simple workaround for this issue is to force all variables to be generated when a model is instantiated. This can be done by adding something like \( \text{eps*sum((i,j),X(i,j))} \) to one of the equations. Note, that \( \text{eps} \) is numerically equal to a 0.0 in GAMS, but has the effect that the term \( \text{sum((i,j),X(i,j))} \) is passed to the solver with 0.0-coefficient.
7.4 Dual Values for PSD constraints

The PSD constraint on a matrix variable $\bar{X}$ is associated with a dual PSD matrix variable $\bar{Y}$. As GAMS is not aware of the PSD constraints, it is also not aware of the corresponding dual variables. Thus, there is no native way to pass the duals for the PSD constraints back to GAMS. (Recall, that the marginals for a PSDMATRIX variable $\bar{X}$ are used to store the dual values associated with the bound constraints on matrix entries.)

To work around this issue, the GAMS/MOSEK interface offers the option `SDPSOLUFILE`. This option allows to specify the name of a GDX file that stores primal values for all entries of a matrix variable and dual values of the corresponding PSD constraint. For a variable $\bar{X}(i,j)$, the GDX file stores the primal matrix value for all entries $(i,j)$ (i.e., not just the ones for which GAMS variables were created, therefor offering another workaround for the issue discussed in the previous section) as level values of variable $\bar{X}$ and the dual matrix for the PSD constraint as marginal values.
1 Introduction

OQNLP and MSNLP are multistart heuristic algorithms designed to find global optima of smooth constrained nonlinear programs (NLPs). By "multistart" we mean that the algorithm calls an NLP solver from multiple starting points, keeps track of all feasible solutions found by that solver, and reports back the best of these as its final solution. The starting points are computed by a scatter search implementation called OptQuest (see www.opttek.com and [Laguna and Marti, 2003]) or by a randomized driver, which generates starting points using probability distributions. There are currently two randomized drivers, Pure Random and Smart Random - see the description of the point_generation parameter. The Smart Random generator uses an initial coarse search to define a promising region within which random starting points are concentrated. With OQNLP, all three drivers are provided, while OptQuest is not present in MSNLP. When interfaced with the GAMS modeling language, any GAMS NLP solver can be called. When used as a callable system, MSNLP uses the LSGRG2 or any GAMS NLP solver (see www.optimalmethods.com and (Smith and Lasdon, 1992)), and this is also provided (optionally) in the GAMS version.

Only the OptQuest driver can handle discrete variables, so OQNLP can attack problems with some or all discrete variables, but MSNLP cannot solve problems with discrete variables. If all variables in a problem are discrete, OQNLP can be applied, but the NLP solver calls play no role, since such solvers vary only the continuous variables, and there aren’t any. Thus the OPTQUEST_ONLY option (see details of options in Section 4) should be used. If a problem is nonsmooth (discontinuous functions and/or derivatives, GAMS problem type DNLP), the NLP solver calls may be less reliable than if the problem was smooth. The OPTQUEST_ONLY option may also be useful in this case.

There is no guarantee that the final solution is a global optimum, and no bound is provided on how far that solution is from the global optimum. However, the algorithm has been tested extensively on 135 problems from the set gathered by Chris Floudas [Floudas, et al., 1999], and it found the best known solution on all but three of them to within a percentage gap of 1%, using default parameters and options (which specify 1000 iterations). It solved two of those three to within 1% given 2000 iterations. It also solved 332 of 339 problems from the GAMS Globallib library to within 1% of the best known solution.
A multistart algorithm can improve the reliability of any NLP solver, by calling it with many starting points. If you have a problem where you think the current NLP solver is failing to find even a local solution, choose an NLP solver and a limit on the number of solver calls, and try OQNLP or MSNLP. Even if a single call to the solver fails, multiple calls from the widely spaced starting points provided by this algorithm have a much better chance of success.

Often an NLP solver fails when it terminates at an infeasible solution. In this situation, the user is not sure if the problem is really infeasible or if the solver is at fault (if all constraints are linear or convex the problem is most likely infeasible). A multistart algorithm can help in such cases. To use it, the problem can be solved in its original form, and some solver calls may terminate with feasible solutions. The algorithm will return the best of these. If all solver calls terminate infeasible, the problem can be reformulated as a feasibility problem. That is, introduce “deviation” or “elastic” variables into each constraint, which measure the amount by which it is violated, and minimize the sum of these violations, ignoring the true objective. OQNLP or MSNLP can be applied to this problem, and either has a much better chance of finding a feasible solution (if one exists) than does a single call to an NLP solver. If no feasible solution is found, you have much more confidence that the problem is truly infeasible.

The OptQuest and randomized drivers generate trial points which are candidate starting points for the NLP solver. These are filtered to provide a smaller subset from which the solver attempts to find a local optimum. In the discussion which follows, we refer to this NLP solver as ”L”, for Local solver.

The most general problem OQNLP can solve has the form

\[ \text{minimize } f(x,y) \quad (17.1) \]

subject to the nonlinear constraints

\[ gl \leq G(x,y) \leq gu \quad (17.2) \]

and the linear constraints

\[ l \leq A_1 x + A_2 y \leq u \quad (17.3) \]

\[ x \in S, \quad y \in Y \quad (17.4) \]

where \( x \) is an \( n \)-dimensional vector of continuous decision variables, \( y \) is a \( p \)-dimensional vector of discrete decision variables, and the vectors \( gl, gu, l, \) and \( u \) contain upper and lower bounds for the nonlinear and linear constraints respectively. The matrices \( A_1 \) and \( A_2 \) are \( m_2 \) by \( n \) and \( m_2 \) by \( p \) respectively, and contain the coefficients of any linear constraints. The set \( S \) is defined by simple bounds on \( x \), and we assume that it is closed and bounded, i.e., that each component of \( x \) has a finite upper and lower bound. This is required by all drivers (see section 4 for a discussion of the parameter ARTIFICIAL_BOUND which provides bounds when none are specified in the model). The set \( Y \) is assumed to be finite, and is often the set of all \( p \)-dimensional binary or integer vectors \( y \). The objective function \( f \) and the \( m_1 \)-dimensional vector of constraint functions \( G \) are assumed to have continuous first partial derivatives at all points in \( S \times Y \). This is necessary so that \( L \) can be applied to the relaxed NLP sub-problems formed from (17.1) - (17.4) by allowing the \( y \) variables to be continuous. The MSNLP system does not allow any discrete variables.

An important function used in this multistart algorithm is the \( L_1 \) exact penalty function, defined as

\[ P_1(x,w) = f(x) + \sum_{i=1}^{m} W_i \text{viol}(g_i(x)) \quad (17.5) \]

where the \( w_i \) are nonnegative penalty weights, \( m = m_1 + m_2 \), and the vector \( g \) has been extended to include the linear constraints (17.4). For simplicity, we assume there are no \( y \) variables: these would be fixed when this function is used. The function \( \text{viol}(g_i(x)) \) is equal to the absolute amount by which the \( i \)th constraint is violated at the point \( x \). It is well known (see [Nash and Sofer, 1996]) that if \( x^* \) is a local optimum of (17.1) - (17.4), \( u^* \) is a corresponding optimal multiplier vector, the second order sufficiency conditions are satisfied at \( (x^*, u^*) \), and

\[ w_i > abs(u_i^*) \quad (17.6) \]

then \( x^* \) is a local unconstrained minimum of \( P_1 \). If (17.1) - (17.4) has several local minima, and each \( w_i \)
is larger than the maximum of all absolute multipliers for constraint \( i \) over all these optima, then \( P_i \) has a local minimum at each of these local constrained minima. We will use \( P_i \) to set thresholds in the merit filter.

### 2 Combining Search Methods and Gradient-Based NLP Solvers

For smooth problems, the relative advantages of a search method over a gradient-based NLP solver are its ability to locate an approximation to a good local solution (often the global optimum), and the fact that it can handle discrete variables. Gradient-based NLP solvers converge to the “nearest” local solution, and have no facilities for discrete variables, unless they are imbedded in a rounding heuristic or branch-and-bound method. Relative disadvantages of search methods are their limited accuracy, and their weak abilities to deal with equality constraints (more generally, narrow feasible regions). They find it difficult to satisfy many nonlinear constraints to high accuracy, but this is a strength of gradient-based NLP solvers. Search methods also require an excessive number of iterations to find approximations to local or global optima accurate to more than two or three significant figures, while gradient-based solvers usually achieve four to eight-digit accuracy rapidly. The motivation for combining search and gradient-based solvers in a multi-start procedure is to achieve the advantages of both while avoiding the disadvantages of either.

### 3 Output

#### 3.1 Log File

When it operates as a GAMS solver, OQNLP and MSNLP will by default write information on their progress to the GAMS log file. When used as a callable system, this information, if requested, will be written to a file opened in the users calling program. The information written consists of:

1. Echos of important configuration and setup values
2. Echo (optionally) of options file settings processed
3. Echos of important algorithm settings, parameters, and termination criteria
4. The iteration log
5. Final results, termination messages, and status report

A segment of that iteration log from stages 1 and 2 of the algorithm is shown below for the problem `ex8_6_2_30.gms`, which is one of a large set of problems described in [Floudas, et al., 1999]. This is a 91 variable unconstrained minimization problem, available from GLOBALLib at www.gamsworld.org/global. There are 200 iterations in stage one and 1000 total iterations (see Appendix A for an algorithm description), with output every 20 iterations and every solver call.

The headings below have the following meanings:

<table>
<thead>
<tr>
<th>Heading</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Itn</td>
<td>iteration number</td>
</tr>
<tr>
<td>Penval</td>
<td>Penalty function value</td>
</tr>
<tr>
<td>Merit Filter</td>
<td>ACC if the merit filter accepts the point, REJ if it rejects</td>
</tr>
<tr>
<td>Merit Threshold</td>
<td>threshold value for merit filter: accepts if Penval &lt; Threshold</td>
</tr>
<tr>
<td>Dist Filter</td>
<td>ACC if the distance filter accepts the point, REJ if it rejects</td>
</tr>
<tr>
<td>Best Obj</td>
<td>Best feasible objective value found thus far</td>
</tr>
</tbody>
</table>
Iterations 0 through 200 below show the initial NLP solver call (at the user-specified initial point, which finds a local minimum with objective value -161.8), and every 20th iteration of stage 1, which has no other solver calls. At iteration 200 stage 1 ends, and the solver is started at the best of the 200 stage 1 points, finding a local min with objective -176.0. The next solver call at iteration 207 finds a better objective of -176.4. Note that, at iteration 207, the OptQuest trial solution has a Penval of -23.18, and this is less than the merit threshold of -20.75, so the merit filter accepts the trial solution, as does the distance filter. The next 9 solver calls fail to improve this value, so Best Obj remains the same, until at iteration 432 a solution with value -176.6 is found. At iteration 473, the solver call finds a value of -177.5. Further solver calls do not find an improved solution and are not shown. The solution with value -177.5 is the best known solution, but OQNLP cannot guarantee this.
The LOCALS file is a text file containing objective and variable values for all local solutions found by MSNLP. It is controlled by the LOCALS_FILE and LOCALS_FILE_FORMAT keywords in the MSNLP Options file. An example for the problem EX_8_1_5 from the Floudas problem set (available on www.gamsworld.org, link to globalworld) is shown below. The headings, included for explanatory purposes and not part of the file, have the following meaning:

<table>
<thead>
<tr>
<th>Heading</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>index of local solution</td>
</tr>
<tr>
<td>Obj</td>
<td>objective value of local solution</td>
</tr>
<tr>
<td>Var</td>
<td>variable index</td>
</tr>
<tr>
<td>Value</td>
<td>variable value</td>
</tr>
</tbody>
</table>

Thus local solutions 1 and 2 both have objective values of -1.03163. The first solution has variable values x = -8.98448e-002, y = 7.12656e-001, where these are in the same order as they are defined in the gams model. The second local solution has x = 8.98418e-002, y = -7.12656e-001. Seven local solutions are found. This output is produced with all default parameter values for MSNLP options and tolerances, except the distance and merit filters were turned off, i.e the keywords USE_DISTANCE_FILTER and USEMerit_FILTER were set to 0 in the MSNLP options file. This causes the NLP solver to be called at every stage 2 trial point, and is recommended if you wish to obtain as many local solutions as possible.

4 The Options File

The options file is a text file containing a set of records, one per line. Each record has the form <keyword> <value>, where the keyword and value are separated by one or more spaces. All relevant options are listed in this guide. You can also get a sample option file with all options and their default values by specifying the single option help in an option file. The list of all options appears in the log file. The options are described below.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>artificial_bound</td>
<td>default upper/lower bound</td>
<td>10000</td>
</tr>
<tr>
<td>basin_decrease_factor</td>
<td>reduction of MAXDIST</td>
<td>0.2</td>
</tr>
<tr>
<td>basin_overlap_fix</td>
<td>switch for MAXDIST logic</td>
<td>1</td>
</tr>
<tr>
<td>distance_factor</td>
<td>distance activation factor</td>
<td>1</td>
</tr>
<tr>
<td>distance_waitcycle</td>
<td>iterations before distance filter threshold is increased</td>
<td>20</td>
</tr>
<tr>
<td>dynamic_distance_filter</td>
<td>switch for MAXDIST reduction logic</td>
<td>1</td>
</tr>
<tr>
<td>dynamicMerit_filter</td>
<td>switch for merit threshold increase logic</td>
<td>1</td>
</tr>
<tr>
<td>enableScreen_output</td>
<td>switch for log output</td>
<td>0</td>
</tr>
<tr>
<td>enableStatistics_log</td>
<td>switch for statistics file stats.log</td>
<td>0</td>
</tr>
<tr>
<td>feasibility_tolerance</td>
<td>feasibility check for point returned by NLP solver</td>
<td>0.0001</td>
</tr>
<tr>
<td>iteration_limit</td>
<td>total number of MSNLP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>iterationPrint_frequency</td>
<td>frequency of iteration print</td>
<td>20</td>
</tr>
<tr>
<td>locals_file</td>
<td>filename for local file</td>
<td></td>
</tr>
<tr>
<td>locals_file_format</td>
<td>file format for local file</td>
<td></td>
</tr>
<tr>
<td>maxtime</td>
<td>maximum runtime in seconds</td>
<td></td>
</tr>
<tr>
<td>maxLocals</td>
<td>maximum number of local optima found</td>
<td>1000</td>
</tr>
<tr>
<td>maxSolverCalls</td>
<td>maximum number of nlp solver calls</td>
<td>1000</td>
</tr>
<tr>
<td>maxSolverCallsNoImprovement</td>
<td>maximum number non-improving solver calls</td>
<td>0</td>
</tr>
<tr>
<td>meritWaitcycle</td>
<td>iterations before merit filter threshold is increased</td>
<td>20</td>
</tr>
<tr>
<td>nlpsolver</td>
<td>NLP solver to be used</td>
<td></td>
</tr>
<tr>
<td>oqnlp_debug</td>
<td>enable debug info</td>
<td>0</td>
</tr>
<tr>
<td>point_generation</td>
<td>starting point generator</td>
<td></td>
</tr>
<tr>
<td>sampling_distribution</td>
<td>distribution for smartrandom1</td>
<td>0</td>
</tr>
<tr>
<td>solvelink</td>
<td>Solvelink for GAMS NLP solver</td>
<td>5</td>
</tr>
<tr>
<td>solverLogToGamsLog</td>
<td>switch to copy the NLP solver log to the normal log file</td>
<td>0</td>
</tr>
<tr>
<td>stage1Iterations</td>
<td>number of iterations in stage 1</td>
<td>200</td>
</tr>
<tr>
<td>thresholdIncreaseFactor</td>
<td>factor to increase merit filter threshold</td>
<td>0.2</td>
</tr>
<tr>
<td>useDistanceFilter</td>
<td>distance filter</td>
<td>1</td>
</tr>
<tr>
<td>useMeritFilter</td>
<td>merit filter</td>
<td>1</td>
</tr>
</tbody>
</table>

**artificial_bound (real):** default upper/lower bound

This value (its negative) is given to the driver as the upper (lower) bound for any variable with no upper or lower bound. However, the original bounds are given to the local solver, so it can produce solutions not limited by this artificial bound. All drivers must have finite upper and lower bounds for each variable. If artificial_bound (or any of the user-supplied bounds) is much larger than any component of the optimal solution, the driver will be less efficient because it is searching over a region that is much larger than needed. Hence the user is advised to try to provide realistic values for all upper and lower bounds. It is even more dangerous to make artificial_bound smaller than some component of a globally optimal solution, since the driver can never
generate a trial point near that solution. It is possible, however, for the local solver to reach a global solution in this case, since the artificial bounds are not imposed on it
(default = 10000)

**basin_decrease_factor (real):** reduction of MAXDIST

This value must be between 0 and 1. If **dynamic_distance_filter** is set to 1, the **MAXDIST** value associated with any local solution is reduced by \( (1 - \text{basin
decrease
dfactor}) \) if **distance_waitcycle** consecutive trial points have distance from that solution less than **MAXDIST**.

Range: \([0, 1]\]
(default = 0.2)

**basin_overlap_fix (integer):** switch for MAXDIST logic

A value of 1 turns on logic which checks the **MAXDIST** values of all pairs of local solutions, and reduces any pair of **MAXDIST** values if their sum is greater than the distance between the 2 solutions. This ensures that the spherical models of their basins of attracting do not overlap. A value of 0 turns off this logic. Turning it off can reduce the number of NLP solver calls, but can also cause the algorithm to miss the global solution.

(default = 1)

**distance_factor (real):** distance activation factor

If the distance between a trial point and any local solution found previously is less than **distance_factor** \( \times \) **MAXDIST**, the NLP solver is not started from that trial point. **MAXDIST** is the largest distance ever traveled to get to that local solution. Increasing **distance_factor** leads to fewer solver calls and risks finding a worse solution. Decreasing it leads to more solver calls and possibly a better solution.

(default = 1)

**distance_waitcycle (integer):** iterations before distance filter threshold is increased

This value must be a positive integer. If the distance filter is used, and there are **distance_waitcycle** consecutive iterations where the distance filter logic causes the NLP solver not to be started, the distance filter threshold is increased by the factor **threshold_increase_factor**. Increasing **distance_waitcycle** usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.

(default = 20)

**dynamic_distance_filter (integer):** switch for MAXDIST reduction logic

A value of 1 turns on logic which reduces the value of **MAXDIST** (described under the **use_distance_filter** keyword) for a local solution if **use_distance_filter** consecutive trial points have a their distances from that solution less than **MAXDIST**. **MAXDIST** is multiplied by \( (1 - \text{basin
development
dfactor}) \). A value of 0 turns off this logic. Turning it off can decrease the number of NLP solver calls, but can also cause the algorithm to miss the global solution.

(default = 1)

**dynamic_merit_filter (integer):** switch for merit threshold increase logic

A value of 1 turns on logic which dynamically varies the parameter which increases the merit filter threshold, **threshold_increase_factor**. If **merit_waitcycle** consecutive trial points have been rejected by the merit filter, this value is replaced by \( \max(\text{threshold_increase_factor}, \text{val}) \), where \( \text{val} \) is the value of **threshold_increase_factor** which causes the merit filter to just accept the best of the previous **merit_waitcycle** trial points. A value of 0 turns off this logic. Turning it off can reduce NLP solver calls, but may lead to a worse final solution.

(default = 1)

**enable_screen_output (integer):** switch for log output

A value of 0 turns off the writing of the iteration log and termination messages to the gams log file that appears on the screen, while 1 enables it.

(default = 0)
**enable_statistics_log** *(integer)*: switch for statistics file stats.log

Using a value of 1 creates a text file called stats.log in the project directory containing one line of problem (name, variables, constraints) and performance information (best objective value, total solver time, iterations, etc) for each problem solved.

(default = 0)

**feasibility_tolerance** *(real)*: feasibility check for point returned by NLP solver

This tolerance is used to check each point returned by an NLP solver for feasibility. If the largest absolute infeasibility at the point is larger than this tolerance, the point is classified infeasible. This test is made because points returned by NLP solvers may occasionally be infeasible despite feasible status codes. Some NLP solvers use internal scaling before testing for feasibility. The unscaled problem may be infeasible, while the scaled one is feasible. If this occurs, increasing this tolerance (to 1.e-2 or larger) often eliminates the problem.

(default = 0.0001)

**iteration_limit** *(integer)*: total number of MSNLP iterations

Increasing this limit can allow MSNLP to find a better solution. Try it if your run using 1000 iterations does not take too long. Surprisingly, the best solution using, say 2000 iterations, may be found in the first 1000 iterations, and that solution may be better than the one found with an iteration limit of 1000. This is because OptQuest changes its search strategy depending on the iteration limit. Because of this, it is also possible that increasing the iteration limit will yield a worse solution, but this is rare. Decreasing this iteration limit usually leads to a worse solution, but also reduces run time. MSNLP iterations can not be set using GAMS iterlim. The GAMS iterlim is used as the iteration limit for the NLP subsolves in a MSNLP run.

(default = 1000)

**iteration_print_frequency** *(integer)*: frequency of iteration print

Synonym: gams_itn_print_frequency

If the MSNLP iteration log is written to the GAMS log file, one line of output is written every \(k\)th iteration, where \(k\) is the value given here.

(default = 20)

**locals_file** *(string)*: filename for local file

Specify a complete path and name for a file to which the objective value and values of all variables for all local solutions found will be written. For example, C:\mydirectory\locals.out. There are 2 possible formats for this file, specified by the **locals_file_format** option below. If there is no locals_file record in the options file, the locals file will not be created.

**locals_file_format** *(string)*: file format for local file

There are 2 possible values for this option. The report entry creates the locals file in a format designed to be examined easily by eye, but processed less easily by a computer program or spreadsheet. The data1 entry creates a file with many records, each on a single line, each line having the following format: [index of local optimum] [objval] [var index] [var value]

(default = report)

   report  Report file format
   data1  Data1 file format

**maxtime** *(integer)*: maximum runtime in seconds

Synonym: reslim

When the execution time exceeds this value, the system will stop, returning the best solution found.

(default = GAMS ResLim)

**maxlocals** *(integer)*: maximum number of local optima found

When the number of distinct local solutions found exceeds the value specified here, the system will stop, returning the best solution found.
max_solver_calls (integer): maximum number of nlp solver calls

When the number of calls to the NLP solver exceeds the value specified here, the system will stop, returning the best solution found.

(default = 1000)

max_solver_calls_noimprovement (integer): maximum number non-improving solver calls

The positive integer specified here will cause the system to stop whenever the number of consecutive solver calls with a fractional improvement in the best objective value found less than 1.e-4 exceeds that value. In other words, if the value specified is 50, and there are more than 50 consecutive solver calls where the relative change in the best objective was less than 1.e-4 in all iterations, the system will stop.

(default = 0)

merit_waitcycle (integer): iterations before merit filter threshold is increased

This value must be a positive integer. If the merit filter is used, and there are merit_waitcycle consecutive iterations where the merit filter logic causes the NLP solver not to be started, the merit filter threshold is increased by the factor threshold_increase_factor. Increasing merit_waitcycle usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.

(default = 20)

nlpsolver (string): NLP solver to be used

This option is available only within GAMS. It specifies the NLP solver to be called. Any GAMS NLP solver for which the user has a license can be used. Further, one can specify an option file for the GAMS NLP solver by appending a .n with n=1..999 to the solver name. For example, NLPSOLVER conopt.1 will instruct the NLP solver CONOPT to use option file conopt.opt, NLPSOLVER conopt.2 will make CONOPT read option file conopt.op2 and so on.

(default = Conopt if licensed otherwise lsgrg)

oqnl Debug (integer): enable debug info

Synonym: msnlp_debug

Values of 1 or 2 cause more information to be written to the iteration log. The default value of 0 suppresses all this output.

Range: [0, 2]

(default = 0)

point_generation (string): starting point generator

Selection of point generation algorithm.

(default = smartrandom1)

  random random point generation Causes trial points to be generated by sampling each variable from a uniform distribution defined within its bounds

  hitandrun hit and run point generation

  smartrandom1 smart random point generation Generates trial points by sampling each variable independently from either normal or triangular distributions, whose parameters are determined as described in Appendix A of the MSNLP User Guide.

  test2 test2 point generation

  test3 test3 point generation

sampling_distribution (integer): distribution for smartrandom1

This keyword is relevant only when point_generation is set to smartrandom1. Then a value of 0 causes normal distributions to be used to generate trial points, while a value of 1 causes triangular distributions to be used.
(default = 0)
  0 normal
  1 triangular

**solvelink (integer):** Solvelink for GAMS NLP solver

(default = 5)
  1 Call GAMS NLP solver via script
  2 Call GAMS NLP solver via module
  5 Call GAMS NLP solver in memory

**solver_log_to_gams_log (integer):** switch to copy the NLP solver log to the normal log file

Setting the parameter to 1 instructs MSNLP to copy the log from the NLP subsolver to the MSNLP log. It can be very helpful to inspect the NLP subsolver log especially if the solver termination code is ???.

(default = 0)

**stage1_iterations (integer):** number of iterations in stage 1

Specifies the total number of iterations in stage 1 of the algorithm, where no NLP solver calls are made. Increasing this sometimes leads to a better starting point for the first local solver call in stage 2, at the cost of delaying that call. Decreasing it can lead to more solver calls, but the first call occurs sooner.

(default = 200)

**threshold_increase_factor (real):** factor to increase merit filter threshold

This value must be nonnegative. If there are merit_waitcycle consecutive MSNLP iterations where the merit filter logic causes the NLP solver not to be called, the merit threshold is increased by multiplying it by (1 + threshold_increase_factor). The same applies to the distance_waitcycle.

(default = 0.2)

**use_distance_filter (integer):** distance filter

Use 0 to turn off the distance filter, the logic which starts the NLP solver at a trial point only if the (Euclidean) distance from that point to any local solution found thus far is greater than the distance threshold. Turning off the distance filter leads to more solver calls and more run time, and increases the chances of finding a global solution. Turn off both distance and merit filters to find (almost) all local solutions.

(default = 1)

**use_merit_filter (integer):** merit filter

Use 0 to turn off the merit filter, the logic which starts the NLP solver at a trial point only if the penalty function value at that point is below the merit threshold. This will lead to more solver calls, but increases the chances of finding a global solution. Turn off both filters if you want to find (almost) all local solutions. This will cause the solver to be called at each stage 2 iteration.

(default = 1)

## 5 Use as a Callable System

MSNLP and OQNLP is also available as a callable system. It currently uses the LSGRG2 or any GAMS NLP solver as its local solver. A sample calling program is provided which a user can easily adapt. The user must provide a C function which computes values of the objective and all constraint functions, given current values of all variables. First partial derivatives of these functions can be approximated by forward or central differences, or may be computed in a user-provided function.
6 Appendix

6.1 Appendix A: Description of the Algorithm

A pseudo-code description of the MSNLP algorithm follows, in which SP \((x_t)\) denotes the starting point generator and \(x_t\) is the candidate starting point produced. We refer to the local NLP solver as \(L(xs, xf)\), where \(xs\) is the starting point and \(xf\) the final point. The function UPDATE LOCALS \((xs, xf, w)\) processes and stores solver output \(xf\), using the starting point \(xs\) to compute the distance from \(xs\) to \(xf\), and produces updated penalty weights, \(w\). For more details, see [Lasdon, Plummer et al., 2004].

MSNLP Algorithm

STAGE 1
\[ x_0 = \text{user initial point} \]
Call \(L(x_0, xf)\)
Call UPDATE LOCALS \((x_0, xf, w)\)
FOR \(i = 1, n_1\) DO
    Call SP \((x_t(i))\)
    Evaluate \(P(x_t(i), w)\)
ENDDO
\(x_t^* = \text{point yielding best value of } P(x_t(i), w) \text{ over all stage one points, } (i = 1, 2, ..., n_1)\).
call \(L(x_t^*, xf)\)
Call UPDATE LOCALS \((x_t^*, xf, w)\)
threshold = \(P(x_t^*, w)\)

STAGE 2
FOR \(i = 1, n_2\) DO
    Call SP \((x_t(i))\)
    Evaluate \(P(x_t(i), w)\)
    Perform merit and distance filter tests:
    Call distance filter( \(x_t(i), dstatus \) )
    Call merit filter( \(x_t(i), \text{threshold, mstatus} \) )
    IF (\(dstatus \text{ and mstatus = } "\text{accept}"\) THEN
        Call \(L(x_t(i), xf)\)
        Call UPDATE LOCALS \((x_t(i), xf, w)\)
    ENDF
ENDDO

After an initial call to \(L\) at the user-provided initial point, \(x_0\), stage 1 of the algorithm performs \(n_1\) iterations in which \(SP(x_t)\) is called, and the L1 exact penalty value \(P(x_t, w)\) is calculated. The user can set \(n_1\) through the MSNLP options file using the STAGE1_ITERATIONS keyword. The point with the smallest of these \(P\) values is chosen as the starting point for the next call to \(L\), which begins stage 2. In this stage, \(n_2\) iterations are performed in which candidate starting points are generated and \(L\) is started at any one which passes the distance and merit filter tests. The options file keyword STAGE2_ITERATIONS sets \(n_2\).

The distance filter helps insure that the starting points for \(L\) are diverse, in the sense that they are not too close to any previously found local solution. Its goal is to prevent \(L\) from starting more than once within the basin of attraction of any local optimum. When a local solution is found, it is stored in a linked list, ordered by its objective value, as is the Euclidean distance between it and the starting point that led to it. If a local solution is located more than once, the maximum of these distances, \(maxdist\), is updated and stored. For each trial point, \(t\), if the distance between \(t\) and any local solution already
found is less than DISTANCE_FACTOR \times maxdist, L is not started from the point, and we obtain the next trial solution from the generator.

This distance filter implicitly assumes that the attraction basins are spherical, with radii at least maxdist. The default value of DISTANCE_FACTOR is 1.0, and it can be set to any positive value in the MSNLP options file—see Section 3. As DISTANCE_FACTOR approaches zero, the filtering effect vanishes, as would be appropriate if there were many closely spaced local solutions. As it becomes larger than 1, the filtering effect increases until eventually L is never started.

The merit filter helps ensure that the starting points for L have high quality, by not starting from candidate points whose exact penalty function value \( P_1 \) (see equation (5), Section 1) is greater than a threshold. This threshold is set initially to the \( P_1 \) value of the best candidate point found in the first stage of the algorithm. If trial points are rejected by this test for more than WAITCYCL consecutive iterations, the threshold is increased by the updating rule:

\[
\text{threshold} \leftarrow \text{threshold} + \text{THRESHOLD\_INCREASE\_FACTOR} \times (1.0 + \text{abs(threshold)})
\]

where the default value of \text{THRESHOLD\_INCREASE\_FACTOR} is 0.2 and that for \text{WAITCYCLE} is 20. The additive 1.0 term is included so that threshold increases by at least \text{THRESHOLD\_INCREASE\_FACTOR} when its current value is near zero. When a trial point is accepted by the merit filter, threshold is decreased by setting it to the \( P_1 \) value of that point.

The combined effect of these 2 filters is that L is started at only a few percent of the trial points, yet global optimal solutions are found for a very high percentage of the test problems. However, the chances of finding a global optimum are increased by increasing \text{ITERATION\_LIMIT} (which we recommend trying first) or by "loosening" either or both filters, although this is rarely necessary in our tests if the dynamic filters and basin overlap fix are used, as they are by default. If the ratio of stage 2 iterations to solver calls is more than 20 using the current filter parameters, and computation times with the default filter parameters are reasonable, you can try loosening the filters. This is achieved for the merit filter either by decreasing \text{WAITCYCLE} or by increasing \text{THRESHOLD\_INCREASE\_FACTOR} (or doing both), and for the distance filter by decreasing DISTANCE_FACTOR. Either or both filters may be turned off, by setting \text{USE\_DISTANCE\_FILTER} and/or \text{USE\_MERIT\_FILTER} to 0. Turning off both causes an NLP solver call at every stage 2 trial point. This is the best way to ensure that all local optima are found, but it can take a long time.

### 6.2 Appendix B: Pure and "Smart" Random Drivers

The "pure" random (PR) driver generates uniformly distributed points within the hyper-rectangle \( S \) defined by the variable bounds. However, this rectangle is often very large, because users often set bounds to \((-\infty, +\infty), (0, +\infty), \) or to large positive and/or negative numbers, particularly in problems with many variables. This usually has little adverse impact on a good local solver, as long as the starting point is chosen well inside the bounds. But the PR generator will often generate starting points with very large absolute component values when some bounds are very large, and this sharply degrades solver performance. Thus we were motivated to develop random generators which control the likelihood of generating candidate points with large components, and intensify the search by focusing points into promising regions. We present two variants, one using normal, the other triangular distributions. Pseudo-code for this "smart random" generator using normal distributions follows, where w is the set of penalty weights determined by the "update locals" logic discussed above, after the first solver call at the user-specified initial point.

**Smart Random Generator with Normal Distributions, SRN \((x_t)\)**

**IF** (first call) **THEN**

Generate \( k_1 \) (default 400) diverse points in \( S \) and evaluate the exact penalty function \( P(x, w) \) at each point.

\[ B = \text{subset of } S \text{ with } k_2 \text{ (default 10) best } P \text{ values} \]

**FOR** \( i = 1, nvars \) **DO**

\[ x_{\text{max}}(i) = \text{max of component } i \text{ of points in } B \]

\[ x_{\text{min}}(i) = \text{min of component } i \text{ of points in } B \]

\[ \mu(i) = (x_{\text{max}}(i) + x_{\text{min}}(i))/2 \]

\[ \text{ratio}(i) = (x_{\text{max}}(i) - x_{\text{min}}(i))/(1+\text{buvar}(i)-\text{blvar}(i)) \]

\[ \text{sigfactor} = 2.0 \]

**IF** (ratio > 0.7) sigfactor = f(ratio)
\[
\text{sigma}(i) = \frac{\text{xmax}(i) - \text{xmin}(i)}{\text{sigfactor}}
\]

ENDDO

ENDIF

FOR \( i = 1, n\text{vars} \) DO

Generate a normally distributed random variable \( rv(i) \) with mean \( \mu(i) \) and standard deviation \( \text{sigma}(i) \)

If \( rv(i) \) is between \( \text{blvar}(i) \) and \( \text{buvar}(i) \), \( xt(i) = rv(i) \)

If \( rv(i) < \text{blvar}(i) \), generate \( xt(i) \) uniformly between \( \text{blvar}(i) \) and \( \text{xmin}(i) \)

If \( rv(i) < \text{buvar}(i) \), generate \( xt(i) \) uniformly between \( \text{xmax}(i) \) and \( \text{buvar}(i) \)

ENDDO

Return \( xt \)

This SRN generator attempts to find a subset, \( B \), of \( k2 \) “good” points, and generates most of its trial points \( xt \), within the smallest rectangle containing \( B \). It first generates a set of \( k1 \) diverse points within the bounds using a stratified random sampling procedure with frequency-based memory. For each variable \( x(i) \), this divides the interval \([\text{blvar}(i), \text{buvar}(i)]\) into 4 equal segments, chooses a segment with probability inversely proportional to the frequency with which it has been chosen thus far, then generates a random point in this segment. We choose \( k2 \) of these points having the best \( P(x, w) \) penalty values, and use the smallest rectangle containing these, intersecting the \( i \)th axis at points \([\text{xmin}(i), \text{xmax}(i)]\), to define \( n \) univariate normal distributions (driver SRN) or \( n \) univariate triangular distributions (driver SRT). The mean of the \( i \)th normal distribution, \( \mu(i) \), is the midpoint of the interval \([\text{xmin}(i), \text{xmax}(i)]\), and this point is also the mode of the \( i \)th triangular distribution, whose lower and upper limits are \( \text{blvar}(i) \) and \( \text{buvar}(i) \). The standard deviation of the \( i \)th normal distribution is selected as described below. The trial point \( xt \) is generated by sampling \( n \) times independently from these distributions. For the driver using normals, if the generated point lies within the bounds, it is accepted. Otherwise, we generate a uniformly distributed point between the violated bound and the start of the interval.

To determine the standard deviation of the normal distributions, we compute \( \text{ratio} \), roughly the ratio of interval width to distance between bounds, where the factor 1.0 is included to avoid division by zero when the bounds are equal (fixed variables). If the interval width is small relative to the distance between bounds for variable \( i \) (\( \text{ratio} \leq 0.7 \)), then the standard deviation \( \text{sigma}(i) \) is half the interval width, so about 1/3 of the \( xt(i) \) values fall outside the interval, providing diversity when the interval does not contain an optimal value for \( x(i) \). If the bounds are large, then \( \text{ratio} \) should be small, say less than 0.1, so \( xt(i) \) values near the bounds are very unlikely. If \( \text{ratio} > 0.7 \), the function \( f \) sets \( \text{sigfactor} \) equal to 2.56 if \( \text{ratio} \) is between 0.7 and 0.8, increasing in steps to 6.2 if \( \text{ratio} > 0.999 \). Thus if \( \text{ratio} \) is near 1.0, more than 99\% of the values fall within the interval, and few have to be projected back within the bounds. The projecting back process avoids undesirable clustering of trial points at a bound, by generating points uniformly between the violated bound and the nearest edge of the interval \([\text{xmin}(i), \text{xmax}(i)]\). When the interval \([\text{xmin}(i), \text{xmax}(i)]\) is sharply skewed toward one of the variable bounds and is much narrower than the distance between the bounds, a symmetric distribution like the normal, combined with our projection procedure, generates too many points between the interval and its nearest bound. A quick scan of the test results indicates that this happens rarely, but an asymmetric distribution like the triangular overcomes this difficulty, and needs no projection.

7 References

1 Introduction

OQNLP and MSNLP are multistart heuristic algorithms designed to find global optima of smooth constrained nonlinear programs (NLPs). By "multistart" we mean that the algorithm calls an NLP solver from multiple starting points, keeps track of all feasible solutions found by that solver, and reports back the best of these as its final solution. The starting points are computed by a scatter search implementation called OptQuest (see www.opttek.com and [Laguna and Marti, 2003]) or by a randomized driver, which generates starting points using probability distributions. There are currently two randomized drivers, Pure Random and Smart Random - see the description of the point_generation parameter. The Smart Random generator uses an initial coarse search to define a promising region within which random starting points are concentrated. With OQNLP, all three drivers are provided, while OptQuest is not present in MSNLP. When interfaced with the GAMS modeling language, any GAMS NLP solver can be called. When used as a callable system, MSNLP uses the LSGRG2 or any GAMS NLP solver (see www.optimalmethods.com and (Smith and Lasdon, 1992)), and this is also provided (optionally) in the GAMS version.

Only the OptQuest driver can handle discrete variables, so OQNLP can attack problems with some or all discrete variables, but MSNLP cannot solve problems with discrete variables. If all variables in a problem are discrete, OQNLP can be applied, but the NLP solver calls play no role, since such solvers vary only the continuous variables, and there aren’t any. Thus the OPTQUEST_ONLY option (see details of options in Section 4) should be used. If a problem is nonsmooth (discontinuous functions and/or derivatives, GAMS problem type DNLP), the NLP solver calls may be less reliable than if the problem was smooth. The OPTQUEST_ONLY option may also be useful in this case.

There is no guarantee that the final solution is a global optimum, and no bound is provided on how far that solution is from the global optimum. However, the algorithm has been tested extensively on 135 problems from the set gathered by Chris Floudas [Floudas, et al., 1999], and it found the best known solution on all but three of them to within a percentage gap of 1%, using default parameters and options (which specify 1000 iterations). It solved two of those three to within 1% given 2000 iterations. It also solved 332 of 339 problems from the GAMS Globallib library to within 1% of the best known solution.
using default parameters, and solved most of the seven remaining ones by increasing the iteration limit or using another NLP solver. These results are described in [Lasdon et al., 2004]. For more information on OQNLP, see [Ugray, et. al., 2003].

A multistart algorithm can improve the reliability of any NLP solver, by calling it with many starting points. If you have a problem where you think the current NLP solver is failing to find even a local solution, choose an NLP solver and a limit on the number of solver calls, and try OQNLP or MSNLP. Even if a single call to the solver fails, multiple calls from the widely spaced starting points provided by this algorithm have a much better chance of success.

Often an NLP solver fails when it terminates at an infeasible solution. In this situation, the user is not sure if the problem is really infeasible or if the solver is at fault (if all constraints are linear or convex the problem is most likely infeasible). A multistart algorithm can help in such cases. To use it, the problem can be solved in its original form, and some solver calls may terminate with feasible solutions. The algorithm will return the best of these. If all solver calls terminate infeasible, the problem can be reformulated as a feasibility problem. That is, introduce "deviation" or "elastic" variables into each constraint, which measure the amount by which it is violated, and minimize the sum of these violations, ignoring the true objective. OQNLP or MSNLP can be applied to this problem, and either has a much better chance of finding a feasible solution (if one exists) than does a single call to an NLP solver. If no feasible solution is found, you have much more confidence that the problem is truly infeasible.

The OptQuest and randomized drivers generate trial points which are candidate starting points for the NLP solver. These are filtered to provide a smaller subset from which the solver attempts to find a local optimum. In the discussion which follows, we refer to this NLP solver as "L", for Local solver.

The most general problem OQNLP can solve has the form

\[
\begin{align*}
\text{minimize} \ f(x,y) \\
\text{subject to} \ g_l(x,y) & \leq gu \\
\text{and the linear constraints} \\
l & \leq A_1x + A_2y \leq u \\
x \in S, \ y \in Y
\end{align*}
\]  

where \( x \) is an \( n \)-dimensional vector of continuous decision variables, \( y \) is a \( p \)-dimensional vector of discrete decision variables, and the vectors \( g_l, gu, l, \) and \( u \) contain upper and lower bounds for the nonlinear and linear constraints respectively. The matrices \( A_1 \) and \( A_2 \) are \( m_2 \) by \( n \) and \( m_2 \) by \( p \) respectively, and contain the coefficients of any linear constraints. The set \( S \) is defined by simple bounds on \( x \), and we assume that it is closed and bounded, i.e., that each component of \( x \) has a finite upper and lower bound. This is required by all drivers (see section 4 for a discussion of the parameter ARTIFICIAL_BOUND which provides bounds when none are specified in the model). The set \( Y \) is assumed to be finite, and is often the set of all \( p \)-dimensional binary or integer vectors \( y \). The objective function \( f \) and the \( m_1 \)-dimensional vector of constraint functions \( G \) are assumed to have continuous first partial derivatives at all points in \( S \times Y \). This is necessary so that \( L \) can be applied to the relaxed NLP sub-problems formed from (18.1) - (18.4) by allowing the \( y \) variables to be continuous. The MSNLP system does not allow any discrete variables.

An important function used in this multistart algorithm is the \( L_1 \) exact penalty function, defined as

\[
P_1(x,w) = f(x) + \sum_{i=1}^{m} W_i \text{viol}(g_i(x))
\]  

where the \( w_i \) are nonnegative penalty weights, \( m = m_1 + m_2 \), and the vector \( g \) has been extended to include the linear constraints (18.4). For simplicity, we assume there are no \( y \) variables: these would be fixed when this function is used. The function \( \text{viol}(g_i(x)) \) is equal to the absolute amount by which the \( i \)th constraint is violated at the point \( x \). It is well known (see [Nash and Sofer, 1996]) that if \( x^* \) is a local optimum of (18.1) - (18.4), \( u^* \) is a corresponding optimal multiplier vector, the second order sufficiency conditions are satisfied at \( (x^*, u^*) \), and

\[
w_i > abs(u_i^*)
\]  

then \( x^* \) is a local unconstrained minimum of \( P_1 \). If (18.1) - (18.4) has several local minima, and each \( w_i \)
is larger than the maximum of all absolute multipliers for constraint \( i \) over all these optima, then \( P_i \) has a local minimum at each of these local constrained minima. We will use \( P_i \) to set thresholds in the merit filter.

2 Combining Search Methods and Gradient-Based NLP Solvers

For smooth problems, the relative advantages of a search method over a gradient-based NLP solver are its ability to locate an approximation to a good local solution (often the global optimum), and the fact that it can handle discrete variables. Gradient-based NLP solvers converge to the "nearest" local solution, and have no facilities for discrete variables, unless they are imbedded in a rounding heuristic or branch-and-bound method. Relative disadvantages of search methods are their limited accuracy, and their weak abilities to deal with equality constraints (more generally, narrow feasible regions). They find it difficult to satisfy many nonlinear constraints to high accuracy, but this is a strength of gradient-based NLP solvers. Search methods also require an excessive number of iterations to find approximations to local or global optima accurate to more than two or three significant figures, while gradient-based solvers usually achieve four to eight-digit accuracy rapidly. The motivation for combining search and gradient-based solvers in a multi-start procedure is to achieve the advantages of both while avoiding the disadvantages of either.

3 Output

3.1 Log File

When it operates as a GAMS solver, OQNLP and MSNLP will by default write information on their progress to the GAMS log file. When used as a callable system, this information, if requested, will be written to a file opened in the users calling program. The information written consists of:

1. Echos of important configuration and setup values

2. Echo (optionally) of options file settings processed

3. Echos of important algorithm settings, parameters, and termination criteria

4. The iteration log

5. Final results, termination messages, and status report

A segment of that iteration log from stages 1 and 2 of the algorithm is shown below for the problem ex8.6.2.30.gms, which is one of a large set of problems described in [Floudas, et al., 1999]. This is a 91 variable unconstrained minimization problem, available from GLOBALLib at www.gamsworld.org/global. There are 200 iterations in stage one and 1000 total iterations (see Appendix A for an algorithm description), with output every 20 iterations and every solver call.

The headings below have the following meanings:

<table>
<thead>
<tr>
<th>Heading</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Itn</td>
<td>iteration number</td>
</tr>
<tr>
<td>Penval</td>
<td>Penalty function value</td>
</tr>
<tr>
<td>Merit Filter</td>
<td>ACC if the merit filter accepts the point, REJ if it rejects</td>
</tr>
<tr>
<td>Merit Threshold</td>
<td>threshold value for merit filter: accepts if Penval &lt; Threshold</td>
</tr>
<tr>
<td>Dist Filter</td>
<td>ACC if the distance filter accepts the point, REJ if it rejects</td>
</tr>
<tr>
<td>Best Obj</td>
<td>Best feasible objective value found thus far</td>
</tr>
</tbody>
</table>
Iterations 0 through 200 below show the initial NLP solver call (at the user-specified initial point, which finds a local minimum with objective value -161.8), and every 20th iteration of stage 1, which has no other solver calls. At iteration 200 stage 1 ends, and the solver is started at the best of the 200 stage 1 points, finding a local min with objective -176.0. The next solver call at iteration 207 finds a better objective of -176.4. Note that, at iteration 207, the OptQuest trial solution has a Penval of -23.18, and this is less than the merit threshold of -20.75, so the merit filter accepts the trial solution, as does the distance filter. The next 9 solver calls fail to improve this value, so Best Obj remains the same, until at iteration 432 a solution with value -176.6 is found. At iteration 473, the solver call finds a value of -177.5. Further solver calls do not find an improved solution and are not shown. The solution with value -177.5 is the best known solution, but OQNLP cannot guarantee this.
3.2 The LOCALS File

The LOCALS file is a text file containing objective and variable values for all local solutions found by MSNLP. It is controlled by the LOCALS_FILE and LOCALS_FILE_FORMAT keywords in the MSNLP Options file. An example for the problem EX_8.1_5 from the Floudas problem set (available on www.gamsworld.org, link to globalworld) is shown below. The headings, included for explanatory purposes and not part of the file, have the following meaning:

<table>
<thead>
<tr>
<th>Heading</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>index of local solution</td>
</tr>
<tr>
<td>Obj</td>
<td>objective value of local solution</td>
</tr>
<tr>
<td>Var</td>
<td>variable index</td>
</tr>
<tr>
<td>Value</td>
<td>variable value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No.</th>
<th>Obj</th>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.03163e+000</td>
<td>1</td>
<td>-8.98448e-002</td>
</tr>
<tr>
<td>1</td>
<td>-1.03163e+000</td>
<td>2</td>
<td>7.12656e-001</td>
</tr>
<tr>
<td>2</td>
<td>-1.03163e+000</td>
<td>1</td>
<td>8.98418e-002</td>
</tr>
<tr>
<td>2</td>
<td>-1.03163e+000</td>
<td>2</td>
<td>-7.12656e-001</td>
</tr>
<tr>
<td>3</td>
<td>-2.15464e-001</td>
<td>1</td>
<td>1.70361e+000</td>
</tr>
<tr>
<td>3</td>
<td>-2.15464e-001</td>
<td>2</td>
<td>-7.96084e-001</td>
</tr>
<tr>
<td>4</td>
<td>-2.15464e-001</td>
<td>1</td>
<td>-1.70361e+000</td>
</tr>
<tr>
<td>4</td>
<td>-2.15464e-001</td>
<td>2</td>
<td>7.96084e-001</td>
</tr>
<tr>
<td>5</td>
<td>0.00000e+000</td>
<td>1</td>
<td>0.00000e+000</td>
</tr>
<tr>
<td>5</td>
<td>0.00000e+000</td>
<td>2</td>
<td>0.00000e+000</td>
</tr>
<tr>
<td>6</td>
<td>2.10425e+000</td>
<td>1</td>
<td>1.60710e+000</td>
</tr>
<tr>
<td>6</td>
<td>2.10425e+000</td>
<td>2</td>
<td>5.68656e-001</td>
</tr>
<tr>
<td>7</td>
<td>2.10425e+000</td>
<td>1</td>
<td>-1.60711e+000</td>
</tr>
<tr>
<td>7</td>
<td>2.10425e+000</td>
<td>2</td>
<td>-5.68651e-001</td>
</tr>
</tbody>
</table>

Thus local solutions 1 and 2 both have objective values of -1.03163. The first solution has variable values $x = -8.98448e-002$, $y = 7.12656e-001$, where these are in the same order as they are defined in the gams model. The second local solution has $x = 8.98418e-002$, $y = -7.12656e-001$. Seven local solutions are found. This output is produced with all default parameter values for MSNLP options and tolerances, except the distance and merit filters were turned off, i.e the keywords USE_DISTANCE_FILTER and USE_MERIT_FILTER were set to 0 in the MSNLP options file. This causes the NLP solver to be called at every stage 2 trial point, and is recommended if you wish to obtain as many local solutions as possible.

4 The Options File

The options file is a text file containing a set of records, one per line. Each record has the form `<keyword> <value>`, where the keyword and value are separated by one or more spaces. All relevant options are listed in this guide. You can also get a sample option file with all options and their default values by specifying the single option `help` in an option file. The list of all options appears in the log file. The options are described below.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>artificial_bound</td>
<td>default upper/lower bound</td>
<td>10000</td>
</tr>
<tr>
<td>basin_decrease_factor</td>
<td>reduction of MAXDIST</td>
<td>0.2</td>
</tr>
<tr>
<td>basin_overlap_fix</td>
<td>switch for MAXDIST logic</td>
<td>1</td>
</tr>
<tr>
<td>distance_factor</td>
<td>distance activation factor</td>
<td>1</td>
</tr>
<tr>
<td>distance_waitcycle</td>
<td>iterations before distance filter threshold is increased</td>
<td>20</td>
</tr>
<tr>
<td>dynamic_distance_filter</td>
<td>switch for MAXDIST reduction logic</td>
<td>1</td>
</tr>
<tr>
<td>dynamic_merit_filter</td>
<td>switch for merit threshold increase logic</td>
<td>1</td>
</tr>
<tr>
<td>enable_screen_output</td>
<td>switch for log output</td>
<td>0</td>
</tr>
<tr>
<td>enable_statistics_log</td>
<td>switch for statistics file stats.log</td>
<td>0</td>
</tr>
<tr>
<td>feasibility_tolerance</td>
<td>feasibility check for point returned by NLP solver</td>
<td>0.0001</td>
</tr>
<tr>
<td>iteration_limit</td>
<td>total number of OQNLP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>iteration_print_frequency</td>
<td>frequency of iteration print</td>
<td>20</td>
</tr>
<tr>
<td>locals_file</td>
<td>filename for local file</td>
<td></td>
</tr>
<tr>
<td>locals_file_format</td>
<td>file format for local file</td>
<td>report</td>
</tr>
<tr>
<td>maxtime</td>
<td>maximum runtime in seconds</td>
<td></td>
</tr>
<tr>
<td>max_locals</td>
<td>maximum number of local optima found</td>
<td></td>
</tr>
<tr>
<td>max_solver_calls</td>
<td>maximum number of nlp solver calls</td>
<td>1000</td>
</tr>
<tr>
<td>max_solver_calls_noimprovement</td>
<td>maximum number non-improving solver calls</td>
<td>0</td>
</tr>
<tr>
<td>merit_waitcycle</td>
<td>iterations before merit filter threshold is increased</td>
<td>20</td>
</tr>
<tr>
<td>nlpsolver</td>
<td>NLP solver to be used</td>
<td>Conopt if licensed otherwise lsgrg</td>
</tr>
<tr>
<td>optquest_only</td>
<td>switch for using local solvers</td>
<td>0</td>
</tr>
<tr>
<td>oqnlp_debug</td>
<td>enable debug info</td>
<td>0</td>
</tr>
<tr>
<td>point_generation</td>
<td>starting point generator</td>
<td>optquest</td>
</tr>
<tr>
<td>sampling_distribution</td>
<td>distribution for smartrandom1</td>
<td>0</td>
</tr>
<tr>
<td>search_type</td>
<td>search strategy</td>
<td>boundary</td>
</tr>
<tr>
<td>solvelink</td>
<td>Solvelink for GAMS NLP solver</td>
<td>5</td>
</tr>
<tr>
<td>solver_log_to_gams_log</td>
<td>switch to copy the NLP solver log to the normal log file</td>
<td>0</td>
</tr>
<tr>
<td>stage1_iterations</td>
<td>number of iterations in stage 1</td>
<td>200</td>
</tr>
<tr>
<td>threshold_increase_factor</td>
<td>factor to increase merit filter threshold</td>
<td>0.2</td>
</tr>
<tr>
<td>use_distance_filter</td>
<td>distance filter</td>
<td>1</td>
</tr>
<tr>
<td>use_linear_constraints</td>
<td>satisfy linear constraints</td>
<td>0</td>
</tr>
<tr>
<td>use_merit_filter</td>
<td>merit filter</td>
<td>1</td>
</tr>
</tbody>
</table>

**artificial_bound (real):** default upper/lower bound

This value (its negative) is given to the driver as the upper (lower) bound for any variable with no upper or lower bound. However, the original bounds are given to the local solver, so it can produce solutions not limited by this artificial bound. All drivers must have finite upper and lower bounds for each variable. If artificial_bound
(or any of the user-supplied bounds) is much larger than any component of the optimal solution, the driver will be less efficient because it is searching over a region that is much larger than needed. Hence the user is advised to try to provide realistic values for all upper and lower bounds. It is even more dangerous to make artificial_bound smaller than some component of a globally optimal solution, since the driver can never generate a trial point near that solution. It is possible, however, for the local solver to reach a global solution in this case, since the artificial bounds are not imposed on it

\( \text{(default = 10000)} \)

\textbf{basin\_decrease\_factor (real): reduction of MAXDIST}

This value must be between 0 and 1. If \text{dynamic\_distance\_filter} is set to 1, the \text{MAXDIST} value associated with any local solution is reduced by \( (1 - \text{basin\_decrease\_factor}) \) if \text{distance\_waitcycle} consecutive trial points have distance from that solution less than \text{MAXDIST}.

Range: \([0, 1]\)

\( \text{(default = 0.2)} \)

\textbf{basin\_overlap\_fix (integer): switch for MAXDIST logic}

A value of 1 turns on logic which checks the \text{MAXDIST} values of all pairs of local solutions, and reduces any pair of \text{MAXDIST} values if their sum is greater than the distance between the 2 solutions. This ensures that the spherical models of their basins of attracting do not overlap. A value of 0 turns off this logic. Turning it off can reduce the number of NLP solver calls, but can also cause the algorithm to miss the global solution.

\( \text{(default = 1)} \)

\textbf{distance\_factor (real): distance activation factor}

If the distance between a trial point and any local solution found previously is less than \text{distance\_factor} * \text{MAXDIST}, the NLP solver is not started from that trial point. \text{MAXDIST} is the largest distance ever traveled to get to that local solution. Increasing \text{distance\_factor} leads to fewer solver calls and risks finding a worse solution. Decreasing it leads to more solver calls and possibly a better solution.

\( \text{(default = 1)} \)

\textbf{distance\_waitcycle (integer): iterations before distance filter threshold is increased}

This value must be a positive integer. If the distance filter is used, and there are \text{distance\_waitcycle} consecutive iterations where the distance filter logic causes the NLP solver not to be started, the distance filter threshold is increased by the factor \text{threshold\_increase\_factor}. Increasing \text{distance\_waitcycle} usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.

\( \text{(default = 20)} \)

\textbf{dynamic\_distance\_filter (integer): switch for MAXDIST reduction logic}

A value of 1 turns on logic which reduces the value of \text{MAXDIST} (described under the \text{use\_distance\_filter} keyword) for a local solution if \text{use\_distance\_filter} consecutive trial points have a their distances from that solution less than \text{MAXDIST}. \text{MAXDIST} is multiplied by \( (1 - \text{basin\_reduction\_factor}) \). A value of 0 turns off this logic. Turning it off can decrease the number of NLP solver calls, but can also lead to a worse final solution.

\( \text{(default = 1)} \)

\textbf{dynamic\_merit\_filter (integer): switch for merit threshold increase logic}

A value of 1 turns on logic which dynamically varies the parameter which increases the merit filter threshold, \text{threshold\_increase\_factor}. If \text{merit\_waitcycle} consecutive trial points have been rejected by the merit filter, this value is replaced by \( \max(\text{threshold\_increase\_factor}, \text{val}) \), where \text{val} is the value of \text{threshold\_increase\_factor} which causes the merit filter to just accept the best of the previous \text{merit\_waitcycle} trial points. A value of 0 turns off this logic. Turning it off can reduce NLP solver calls, but may lead to a worse final solution.

\( \text{(default = 1)} \)

\textbf{enable\_screen\_output (integer): switch for log output}
A value of 0 turns off the writing of the iteration log and termination messages to the gams log file that appears on the screen, while 1 enables it.

(default = 0)

`enable_statistics_log (integer):` switch for statistics file stats.log

Using a value of 1 creates a text file called stats.log in the project directory containing one line of problem (name, variables, constraints) and performance information (best objective value, total solver time, iterations, iterations to best solution, etc) for each problem solved.

(default = 0)

`feasibility_tolerance (real):` feasibility check for point returned by NLP solver

This tolerance is used to check each point returned by an NLP solver for feasibility. If the largest absolute infeasibility at the point is larger than this tolerance, the point is classified infeasible. This test is made because points returned by NLP solvers may occasionally be infeasible despite feasible status codes. Some NLP solvers use internal scaling before testing for feasibility. The unscaled problem may be infeasible, while the scaled one is feasible. If this occurs, increasing this tolerance (to 1.e-2 or larger) often eliminates the problem.

(default = 0.0001)

`iteration_limit (integer):` total number of OQNLP iterations

Increasing this limit can allow OQNLP to find a better solution. Try it if your run using 1000 iterations does not take too long. Surprisingly, the best solution using, say 2000 iterations, may be found in the first 1000 iterations, and that solution may be better than the one found with an iteration limit of 1000. This is because OptQuest changes its search strategy depending on the iteration limit. Because of this, it is also possible that increasing the iteration limit will yield a worse solution, but this is rare. Decreasing this iteration limit usually leads to a worse solution, but also reduces run time. OQNLP iterations can not be set using GAMS iterlim. The GAMS iterlim is used as the iteration limit for the NLP subsolves in a OQNLP run.

(default = 1000)

`iteration_print_frequency (integer):` frequency of iteration print

Synonym: gams_itn_print_frequency

If the OQNLP iteration log is written to the GAMS log file, one line of output is written every \( k \)th iteration, where \( k \) is the value given here.

(default = 20)

`locals_file (string):` filename for local file

Specify a complete path and name for a file to which the objective value and values of all variables for all local solutions found will be written. For example, `C:\mydirectory\locals.out`. There are 2 possible formats for this file, specified by the `locals_file_format` option below. If there is no `locals_file` record in the options file, the locals file will not be created.

`locals_file_format (string):` file format for local file

There are 2 possible values for this option. The `report` entry creates the locals file in a format designed to be examined easily by eye, but processed less easily by a computer program or spreadsheet. The `data1` entry creates a file with many records, each on a single line, each line having the following format: [index of local optimum] [objval] [var index] [var value]

(default = `report`)

- `report` Report file format
- `data1` Data1 file format

`maxtime (integer):` maximum runtime in seconds

Synonym: reslim

When the execution time exceeds this value, the system will stop, returning the best solution found.

(default = GAMS ResLim)
**max.locals** *(integer)*: maximum number of local optima found

When the number of distinct local solutions found exceeds the value specified here, the system will stop, returning the best solution found.

(default = 1000)

**max.solver.calls** *(integer)*: maximum number of NLP solver calls

When the number of calls to the NLP solver exceeds the value specified here, the system will stop, returning the best solution found.

(default = 1000)

**max.solver.calls.noimprovement** *(integer)*: maximum number non-improving solver calls

The positive integer specified here will cause the system to stop whenever the number of consecutive solver calls with a fractional improvement in the best objective value found less than 1.e-4 exceeds that value. In other words, if the value specified is 50, and there are more than 50 consecutive solver calls where the relative change in the best objective was less than 1.e-4 in all iterations, the system will stop.

(default = 0)

**merit.waitcycle** *(integer)*: iterations before merit filter threshold is increased

This value must be a positive integer. If the merit filter is used, and there are merit.waitcycle consecutive iterations where the merit filter logic causes the NLP solver not to be started, the merit filter threshold is increased by the factor threshold.increase.factor. Increasing merit.waitcycle usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.

(default = 20)

**nlpsolver** *(string)*: NLP solver to be used

This option is available only within GAMS. It specifies the NLP solver to be called. Any GAMS NLP solver for which the user has a license can be used. Further, one can specify an option file for the GAMS NLP solver by appending a .n with n=1..999 to the solver name. For example, NLPSOLVER conopt.1 will instruct the NLP solver CONOPT to use option file conopt.opt, NLPSOLVER conopt.2 will make CONOPT read option file conopt.op2 and so on.

(default = Conopt if licensed otherwise lsgrg)

**optquest_only** *(integer)*: switch for using local solvers

This option applies only to the OptQuest driver. If you think the NLP solver is taking too long and/or not working well, choosing 1 will stop it from being called. This may occur if the problem is of type DNLP, where one or more problem functions are discontinuous or have discontinuous derivatives. If the problem has only discrete (integer) variables, choose 1, as there is nothing for the NLP solver to do (since it optimizes over the continuous variables when the integers are fixed, and there none).

(default = 0)

0 use local solver

1 no local solver calls

**oqnlp.debug** *(integer)*: enable debug info

Synonym: msnlp.debug

Values of 1 or 2 cause more information to be written to the iteration log. The default value of 0 suppresses all this output.

Range: [0, 2]

(default = 0)

**point_generation** *(string)*: starting point generator

Selection of point generation algorithm.

(default = optquest)
random point generation Causes trial points to be generated by sampling each variable from a uniform distribution defined within its bounds

hit and run point generation

smart random point generation Generates trial points by sampling each variable independently from either normal or triangular distributions, whose parameters are determined as described in Appendix A of the OQNLP User Guide.

test2 point generation

test3 point generation

OptQuest generates starting point Causes trial points to be generated by the OptQuest driver

**sampling_distribution** *(integer):* distribution for smart random

This keyword is relevant only when **point_generation** is set to smart random. Then a value of 0 causes normal distributions to be used to generate trial points, while a value of 1 causes triangular distributions to be used.

(default = 0)

0 normal

1 triangular

**search_type** *(string):* search strategy

This option applies only to the OptQuest driver, and controls the search strategy used by OptQuest. There are three choices that are relevant for use within OQNLP.

(default = boundary)

aggressive aggressive search This choice controls the population update in step 7 of the OptQuest algorithm (see Appendix A of the User Guide). It triggers a very aggressive update, which keeps the best of the points generated from the current population as the new population. The risk in this is that all points in the new population may cluster in a small portion of the search volume, and regions far from this volume will not be explored in the next cycle.

boundary boundary search This option affects the trial points generated by OptQuest, directing them toward the boundary of the region defined by the linear constraints and variable bounds.

crossover crossover search This option affects how OptQuest trial points are generated from population points. It retains the linear combination operator discussed in Appendix A of the User Guide, but adds a crossover operator, similar to those used in evolutionary or genetic algorithms, to create 2 additional trial points.

**solvelink** *(integer):* Solvelink for GAMS NLP solver

(default = 5)

1 Call GAMS NLP solver via script

2 Call GAMS NLP solver via module

5 Call GAMS NLP solver in memory

**solver_log_to_gams_log** *(integer):* switch to copy the NLP solver log to the normal log file

Setting the parameter to 1 instructs OQNLP to copy the log from the NLP subsolver to the OQNLP log. It can be very helpful to inspect the NLP subsolver log especially if the solver termination code is ???.

(default = 0)

**stage1_iterations** *(integer):* number of iterations in stage 1

Specifies the total number of iterations in stage 1 of the algorithm, where no NLP solver calls are made. Increasing this sometimes leads to a better starting point for the first local solver call in stage 2, at the cost of delaying that call. Decreasing it can lead to more solver calls, but the first call occurs sooner.

(default = 200)
threshold_increase_factor (real): factor to increase merit filter threshold

This value must be nonnegative. If there are merit_waitcycle consecutive OQNLP iterations where the merit filter logic causes the NLP solver not to be called, the merit threshold is increased by multiplying it by (1+threshold_increase_factor). The same applies to the distance_waitcycle.

(default = 0.2)

use_distance_filter (integer): distance filter

Use 0 to turn off the distance filter, the logic which starts the NLP solver at a trial point only if the (Euclidean) distance from that point to any local solution found thus far is greater than the distance threshold. Turning off the distance filter leads to more solver calls and more run time, and increases the chances of finding a global solution. Turn off both distance and merit filters to find (almost) all local solutions.

(default = 1)

use_linear_constraints (integer): satisfy linear constraints

This option applies only to the OptQuest driver, and to problems that have linear constraints other than simple bounds on the variables. Using 1 (all OptQuest trial points satisfy the linear constraints) often leads to fewer iterations and solver calls, but OptQuest has to solve an LP to project each trial point onto the linear constraints. For large problems (more than 100 variables), this can greatly increase run time, so the default value is off (0).

(default = 0)

use_merit_filter (integer): merit filter

Use 0 to turn off the merit filter, the logic which starts the NLP solver at a trial point only if the penalty function value at that point is below the merit threshold. This will lead to more solver calls, but increases the chances of finding a global solution. Turn off both filters if you want to find (almost) all local solutions. This will cause the solver to be called at each stage 2 iteration.

(default = 1)

5 Use as a Callable System

MSNLP and OQNLP is also available as a callable system. It currently uses the LSGRG2 or any GAMS NLP solver as its local solver. A sample calling program is provided which a user can easily adapt. The user must provide a C function which computes values of the objective and all constraint functions, given current values of all variables. First partial derivatives of these functions can be approximated by forward or central differences, or may be computed in a user-provided function.

6 Appendix

6.1 Appendix A: Description of the Algorithm

A pseudo-code description of the MSNLP algorithm follows, in which SP (xt) denotes the starting point generator and xt is the candidate starting point produced. We refer to the local NLP solver as \( L(xs,xf) \), where \( xs \) is the starting point and \( xf \) the final point. The function \( \text{UPDATE LOCALS} (xs,xf,w) \) processes and stores solver output \( xf \), using the starting point \( xs \) to compute the distance from \( xs \) to \( xf \), and produces updated penalty weights, \( w \). For more details, see [Lasdon, Plummer et al., 2004].

MSNLP Algorithm

STAGE 1

\( x_0 = \) user initial point

Call \( L(x_0,xf) \)

Call \( \text{UPDATE LOCALS} (x_0,xf,w) \)

FOR \( i = 1,n1 \) DO
Call SP \((x_t(i))\)
Evaluate \(P(x_t(i),w)\)

ENDO

\(x_t^* = \text{point yielding best value of } P(x_t(i),w) \text{ over all stage one points, } (i = 1,2,...,n1)\).

call \(L(x_t^*, x_f)\)

Call UPDATE LOCALS \((x_t^*, x_f, w)\)

threshold = \(P(x_t^*, w)\)

STAGE 2

FOR \(i = 1, n2 \) DO

Call SP \((x_t(i))\)
Evaluate \(P(x_t(i),w)\)

Perform merit and distance filter tests:
Call distance filter\((x_t(i), dstatus)\)
Call merit filter\((x_t(i), threshold, mstatus)\)

IF \((dstatus \text{ and } mstatus = "accept")\) THEN

Call \(L(x_t(i), x_f)\)

Call UPDATE LOCALS \((x_t(i), x_f, w)\)

ENDIF

ENDDO

After an initial call to \(L\) at the user-provided initial point, \(x_0\), stage 1 of the algorithm performs \(n1\) iterations in which SP \((x_t)\) is called, and the L1 exact penalty value \(P(x_t, w)\) is calculated. The user can set \(n1\) through the MSNLP options file using the \texttt{STAGE1\_ITERATIONS} keyword. The point with the smallest of these \(P\) values is chosen as the starting point for the next call to \(L\), which begins stage 2. In this stage, \(n2\) iterations are performed in which candidate starting points are generated and \(L\) is started at any one which passes the distance and merit filter tests. The options file keyword \texttt{STAGE2\_ITERATIONS} sets \(n2\).

The distance filter helps ensure that the starting points for \(L\) are diverse, in the sense that they are not too close to any previously found local solution. Its goal is to prevent \(L\) from starting more than once within the basin of attraction of any local optimum. When a local solution is found, it is stored in a linked list, ordered by its objective value, as is the Euclidean distance between it and the starting point that led to it. If a local solution is located more than once, the maximum of these distances, \(\text{maxdist}\), is updated and stored. For each trial point, \(t\), if the distance between \(t\) and any local solution already found is less than \(\text{DISTANCE\_FACTOR* maxdist}\), \(L\) is not started from the point, and we obtain the next trial solution from the generator.

This distance filter implicitly assumes that the attraction basins are spherical, with radii at least \(\text{maxdist}\). The default value of \texttt{DISTANCE\_FACTOR} is 1.0, and it can be set to any positive value in the MSNLP options file—see Section 3. As \texttt{DISTANCE\_FACTOR} approaches zero, the filtering effect vanishes, as would be appropriate if there were many closely spaced local solutions. As it becomes larger than 1, the filtering effect increases until eventually \(L\) is never started.

The merit filter helps ensure that the starting points for \(L\) have high quality, by not starting from candidate points whose exact penalty function value \(P_1\) (see equation (5), Section 1) is greater than a threshold. This threshold is set initially to the \(P_1\) value of the best candidate point found in the first stage of the algorithm. If trial points are rejected by this test for more than \(\text{WAITCYCLE}\) consecutive iterations, the threshold is increased by the updating rule:

\[
\text{threshold} \leftarrow \text{threshold} + \text{THRESHOLD\_INCREASE\_FACTOR} \times (1.0 + \text{abs(threshold)})
\]

where the default value of \texttt{THRESHOLD\_INCREASE\_FACTOR} is 0.2 and that for \texttt{WAITCYCLE} is 20. The additive 1.0 term is included so that threshold increases by at least \texttt{THRESHOLD\_INCREASE\_FACTOR} when its current value is near zero. When a trial point is accepted by the merit filter, threshold is decreased by setting it to the \(P_1\) value of that point.

The combined effect of these 2 filters is that \(L\) is started at only a few percent of the trial points, yet global optimal solutions are found for a very high percentage of the test problems. However, the chances of finding a global optimum are increased by increasing \texttt{ITERATION\_LIMIT} (which we recommend trying first) or by “loosening” either or both filters, although this
is rarely necessary in our tests if the dynamic filters and basin overlap fix are used, as they are by default. If the ratio of stage 2 iterations to solver calls is more than 20 using the current filter parameters, and computation times with the default filter parameters are reasonable, you can try loosening the filters. This is achieved for the merit filter either by decreasing \texttt{WAITCYCLE} or by increasing \texttt{THRESHOLD\_INCREASE\_FACTOR} (or doing both), and for the distance filter by decreasing \texttt{DISTANCE\_FACTOR}. Either or both filters may be turned off, by setting \texttt{USE\_DISTANCE\_FILTER} and/or \texttt{USE\_MERIT\_FILTER} to 0. Turning off both causes an NLP solver call at every stage 2 trial point. This is the best way to insure that all local optima are found, but it can take a long time.

6.2 Appendix B: Pure and ”Smart” Random Drivers

The ”pure” random (PR) driver generates uniformly distributed points within the hyper-rectangle \( S \) defined by the variable bounds. However, this rectangle is often very large, because users often set bounds to \((-\infty, +\infty), (0, +\infty), \) or to large positive and/or negative numbers, particularly in problems with many variables. This usually has little adverse impact on a good local solver, as long as the starting point is chosen well inside the bounds. But the PR generator will often generate starting points with very large absolute component values when some bounds are very large, and this sharply degrades solver performance. Thus we were motivated to develop random generators which control the likelihood of generating candidate points with large components, and intensify the search by focusing points into promising regions. We present two variants, one using normal, the other triangular distributions. Pseudo-code for this ”smart random” generator using normal distributions follows, where \( w \) is the set of penalty weights determined by the ”update locals” logic discussed above, after the first solver call at the user-specified initial point.

Smart Random Generator with Normal Distributions, SRN \((x_t)\)

IF (first call) THEN

Generate \( k_1 \) (default 400) diverse points in \( S \) and evaluate the exact penalty function \( P(x, w) \) at each point.

\( B = \) subset of \( S \) with \( k_2 \) (default 10) best \( P \) values

FOR \( i = 1, nvars \) DO

\( x_{\text{max}}(i) = \) max of component \( i \) of points in \( B \)

\( x_{\text{min}}(i) = \) min of component \( i \) of points in \( B \)

\( \mu(i) = (x_{\text{max}}(i) + x_{\text{min}}(i))/2 \)

\( \text{ratio}(i) = (x_{\text{max}}(i) - x_{\text{min}}(i))/(1 + \text{buvar}(i) - \text{blvar}(i)) \)

\( \text{sigfactor} = 2.0 \)

IF (\( \text{ratio} > 0.7 \)) \( \text{sigfactor} = f(\text{ratio}) \)

\( \sigma(i) = (x_{\text{max}}(i) - x_{\text{min}}(i))/\text{sigfactor} \)

ENDDO

ENDIF

FOR \( i = 1, nvars \) DO

Generate a normally distributed random variable \( r_v(i) \) with mean \( \mu(i) \) and standard deviation \( \sigma(i) \)

If \( r_v(i) \) is between \( \text{blvar}(i) \) and \( \text{buvar}(i) \), \( x_t(i) = r_v(i) \)

If \( r_v(i) < \text{blvar}(i) \), generate \( x_t(i) \) uniformly between \( \text{blvar}(i) \) and \( x_{\text{min}}(i) \)

If \( r_v(i) > \text{buvar}(i) \), generate \( x_t(i) \) uniformly between \( x_{\text{max}}(i) \) and \( \text{buvar}(i) \)

ENDDO

Return \( x_t \)

This SRN generator attempts to find a subset, \( B \), of \( k_2 \) ”good” points, and generates most of its trial points \( x_t \), within the smallest rectangle containing \( B \). It first generates a set of \( k_1 \) diverse points within the bounds using a stratified random sampling procedure with frequency-based memory. For each variable \( x(i) \), this divides the interval \([\text{blvar}(i), \text{buvar}(i)]\) into 4 equal segments, chooses a segment with probability inversely proportional to the frequency with which it has been chosen thus far, then generates a random point in this segment. We choose \( k_2 \) of these points having the best \( P(x, w) \) penalty
values, and use the smallest rectangle containing these, intersecting the ith axis at points \([x_{\text{min}}(i), x_{\text{max}}(i)]\), to define \(n\) univariate normal distributions (driver SRN) or \(n\) univariate triangular distributions (driver SRT). The mean of the ith normal distribution, \(\mu(i)\), is the midpoint of the interval \([x_{\text{min}}(i), x_{\text{max}}(i)]\), and this point is also the mode of the ith triangular distribution, whose lower and upper limits are \(blvar(i)\) and \(buvar(i)\). The standard deviation of the ith normal distribution is selected as described below. The trial point \(x_t\) is generated by sampling \(n\) times independently from these distributions. For the driver using normals, if the generated point lies within the bounds, it is accepted. Otherwise, we generate a uniformly distributed point between the violated bound and the start of the interval.

To determine the standard deviation of the normal distributions, we compute \(\text{ratio}\), roughly the ratio of interval width to distance between bounds, where the factor 1.0 is included to avoid division by zero when the bounds are equal (fixed variables). If the interval width is small relative to the distance between bounds for variable \(i\) \((\text{ratio} \leq 0.7)\), then the standard deviation \(\sigma(i)\) is half the interval width, so about 1/3 of the \(x_t(i)\) values fall outside the interval, providing diversity when the interval does not contain an optimal value for \(x(i)\). If the bounds are large, then ratio should be small, say less than 0.1, so \(x_t(i)\) values near the bounds are very unlikely. If \(\text{ratio} > 0.7\), the function \(f\) sets sigfactor equal to 2.56 if ratio is between 0.7 and 0.8, increasing in steps to 6.2 if \(\text{ratio} > 0.999\). Thus if \(\text{ratio}\) is near 1.0, more than 99% of the values fall within the interval, and few have to be projected back within the bounds. The projecting back process avoids undesirable clustering of trial points at a bound, by generating points uniformly between the violated bound and the nearest edge of the interval \([x_{\text{min}}(i), x_{\text{max}}(i)]\). When the interval \([x_{\text{min}}(i), x_{\text{max}}(i)]\) is sharply skewed toward one of the variable bounds and is much narrower than the distance between the bounds, a symmetric distribution like the normal, combined with our projection procedure, generates too many points between the interval and its nearest bound. A quick scan of the test results indicates that this happens rarely, but an asymmetric distribution like the triangular overcomes this difficulty, and needs no projection.

7 References


A fundamental problem of mathematics is to find a solution to a square system of nonlinear equations. Two generalizations of nonlinear equations have been developed, a constrained nonlinear system which incorporates bounds on the variables, and the complementarity problem (MCP). This document is primarily concerned with the complementarity problem.

The PATH solver for MCP models is a Newton-based solver that combines a number of the most effective variations, extensions, and enhancements of this powerful technique. See PATH vs MILES for a comparison with MILES. Algorithmic details can also be found in papers and technical reports by Dirkse, Ferris, and Munson on Ferris’ Home Page.

The complementarity problem adds a combinatorial twist to the classic square system of nonlinear equations, thus enabling a broader range of situations to be modeled. In its simplest form, the combinatorial problem is to choose from $2^n$ inequalities a subset of $n$ that will be satisfied as equations. These problems arise in a variety of disciplines including engineering and
economics [18] where we might want to compute Wardropian and Walrasian equilibria, and optimization where we can model the first order optimality conditions for nonlinear programs [29, 30]. Other examples, such as bimatrix games [31] and options pricing [27], abound.

Our development of complementarity is done by example. We begin by looking at the optimality conditions for a transportation problem and some extensions leading to the nonlinear complementarity problem. We then discuss a Walrasian equilibrium model and use it to motivate the more general mixed complementarity problem. We conclude this chapter with information on solving the models using the PATH solver and interpreting the results.

1.1 Transportation Problem

The transportation model is a linear program where demand for a single commodity must be satisfied by suppliers at minimal transportation cost. The underlying transportation network is given as a set $\mathcal{A}$ of arcs, where $(i, j) \in \mathcal{A}$ means that there is a route from supplier $i$ to demand center $j$. The problem variables are the quantities $x_{i,j}$ shipped over each arc $(i, j) \in \mathcal{A}$. The linear program can be written mathematically as

$$
\begin{align*}
\min_{x} & \sum_{(i, j) \in \mathcal{A}} c_{i,j} x_{i,j} \\
\text{subject to} & \sum_{(i, j) \in \mathcal{A}} x_{i,j} \leq s_i, \forall i \\
& \sum_{(i, j) \in \mathcal{A}} x_{i,j} \geq d_j, \forall j.
\end{align*}
$$

where $c_{i,j}$ is the unit shipment cost on the arc $(i, j)$, $s_i$ is the available supply at $i$, and $d_j$ is the demand at $j$.

The derivation of the optimality conditions for this linear program begins by associating with each constraint a multiplier, alternatively termed a dual variable or shadow price. These multipliers represent the marginal price on changes to the corresponding constraint. We label the prices on the supply constraint $p^s$ and those on the demand constraint $p^d$. Intuitively, for each supply node $i$

$$0 \leq p_i^s, \quad s_i \geq \sum_{j: (i, j) \in \mathcal{A}} x_{i,j}.$$ 

Consider the case when $s_i > \sum_{j: (i, j) \in \mathcal{A}} x_{i,j}$, that is there is excess supply at $i$. Then, in a competitive marketplace, no rational person is willing to pay for more supply at node $i$; it is already over-supplied. Therefore, $p_i^s = 0$. Alternatively, when $s_i = \sum_{j: (i, j) \in \mathcal{A}} x_{i,j}$, that is node $i$ clears, we might be willing to pay for additional supply of the good. Therefore, $p_i^s \geq 0$. We write these two conditions succinctly as:

$$0 \leq p_i^s \perp s_i \geq \sum_{j: (i, j) \in \mathcal{A}} x_{i,j}, \quad \forall i$$

where the $\perp$ notation is understood to mean that at least one of the adjacent inequalities must be satisfied as an equality. For example, either $0 = p_i^s$, the first case, or $s_i = \sum_{j: (i, j) \in \mathcal{A}} x_{i,j}$, the second case.

Similarly, at each node $j$, the demand must be satisfied in any feasible solution, that is

$$\sum_{i: (i, j) \in \mathcal{A}} x_{i,j} \geq d_j.$$ 

Furthermore, the model assumes all prices are nonnegative, $0 \leq p_j^d$. If there is too much of the commodity supplied, $\sum_{i: (i, j) \in \mathcal{A}} x_{i,j} > d_j$, then, in a competitive marketplace, the price $p_j^d$ will be driven down to 0. Summing these relationships gives the following complementarity condition:

$$0 \leq p_j^d \perp \sum_{i: (i, j) \in \mathcal{A}} x_{i,j} \geq d_j, \quad \forall j.$$ 

The supply price at $i$ plus the transportation cost $c_{i,j}$ from $i$ to $j$ must exceed the market price at $j$. That is, $p_i^s + c_{i,j} \geq p_j^d$. Otherwise, in a competitive marketplace, another producer will replicate supplier $i$ increasing the supply of the good in question which drives down the market price. This chain would repeat until the inequality is satisfied. Furthermore, if the cost of delivery strictly exceeds the market price, that is $p_i^s + c_{i,j} > p_j^d$, then nothing is shipped from $i$ to $j$ because doing so would incur a loss and $x_{i,j} = 0$. Therefore,

$$0 \leq x_{i,j} \perp p_i^s + c_{i,j} \geq p_j^d, \quad \forall (i, j) \in \mathcal{A}.$$
We combine the three conditions into a single problem,

\[
\begin{align*}
0 \leq p_i^s & \perp s_i \geq \sum_{j \in \mathcal{A} \setminus \mathcal{B}} x_{i,j}, \quad \forall i \\
0 \leq p_j^d & \perp \sum_{i \in \mathcal{A} \setminus \mathcal{B}} x_{i,j} \geq d_j, \quad \forall j \\
0 \leq x_{i,j} & \perp p_i^s + c_{i,j} \geq p_j^d, \quad \forall (i,j) \in \mathcal{A}.
\end{align*}
\]  

(2)

This model defines a linear complementarity problem that is easily recognized as the complementary slackness conditions [6] of the linear program (1). For linear programs the complementary slackness conditions are both necessary and sufficient for \( x \) to be an optimal solution of the problem (1). Furthermore, the conditions (2) are also the necessary and sufficient optimality conditions for a related problem in the variables \( (p^s, p^d) \)

\[
\max_{p^s, p^d \geq 0} \quad \sum_j d_j p_j^d - \sum_i s_i p_i^s \\
\text{subject to} \quad c_{i,j} \geq p_j^d - p_i^s, \quad \forall (i,j) \in \mathcal{A}
\]

termed the dual linear program (hence the nomenclature "dual variables").

Looking at (2) a bit more closely we can gain further insight into complementarity problems. A solution of (2) tells us the arcs used to transport goods. A priori we do not need to specify which arcs to use, the solution itself indicates them. This property represents the key contribution of a complementarity problem over a system of equations. If we know what arcs to send flow down, we can just solve a simple system of linear equations. However, the key to the modeling power of complementarity is that it chooses which of the inequalities in (2) to satisfy as equations. In economics we can use this property to generate a model with different regimes and let the solution determine which ones are active. A regime shift could, for example, be a back stop technology like windmills that become profitable if a \( CO_2 \) tax is increased.

**GAMS Code**

The GAMS code for the complementarity version of the transportation problem is given in Figure 1.; the actual data for the model is assumed to be given in the file transmcp.dat. Note that the model written corresponds very closely to (2). In GAMS, the \( \perp \) sign is replaced in the model statement with a ".". It is precisely at this point that the pairing of variables and equations shown in (2) occurs in the GAMS code. For example, the function defined by rational is complementary to the variable \( x \). To inform a solver of the bounds, the standard GAMS statements on the variables can be used, namely (for a declared variable \( z(i) \)):

\[
z.\text{lo}(i) = 0;
\]

or alternatively

\[
\text{positive variable } z;
\]

Further information on the GAMS syntax can be found in [35]. *Note that GAMS requires the modeler to write } F(z) \geq 0 \text{ whenever the complementary variable is lower bounded, and does not allow the alternative form } 0 \leq F(z).*

**Figure 1:** A simple MCP model in GAMS, transmcp.gms

```gams
sets i canning plants,
    j markets ;

parameter
    s(i) capacity of plant i in cases,
    d(j) demand at market j in cases,
    c(i,j) transport cost in thousands of dollars per case ;

$include transmcp.dat

positive variables
    x(i,j) shipment quantities in cases
    p_demand(j) price at market j
```

p_supply(i)  price at plant i;
equations
  supply(i)  observe supply limit at plant i
  demand(j) satisfy demand at market j
  rational(i,j);  
supply(i) .. s(i) =g= sum(j, x(i,j)) ; 
demand(j) .. sum(i, x(i,j)) =g= d(j) ;
rational(i,j) .. p_supply(i) + c(i,j) =g= p_demand(j) ;
model transport / rational.x, demand.p_demand, supply.p_supply /;
solve transport using mcp;

Extension: Model Generalization

While many interior point methods for linear programming exploit this complementarity framework (so-called primal-dual methods [37]), the real power of this modeling format is the new problem instances it enables a modeler to create. We now show some examples of how to extend the simple model (2) to investigate other issues and facets of the problem at hand.

Demand in the model of Figure 1 is independent of the prices \( p \). Since the prices \( p \) are variables in the complementarity problem (2), we can easily replace the constant demand \( d \) with a function \( d(p) \) in the complementarity setting. Clearly, any algebraic function of \( p \) that can be expressed in GAMS can now be added to the model given in Figure 1. For example, a linear demand function could be expressed using

\[
\sum_{i \in \mathcal{A}} x_{i,j} \geq d_j(1 - p^d_j), \forall j.
\]

Note that the demand is rather strange if \( p^d_j \) exceeds 1. Other more reasonable examples for \( d(p) \) are easily derived from Cobb-Douglas or CES utilities. For those examples, the resulting complementarity problem becomes nonlinear in the variables \( p \). Details of complementarity for more general transportation models can be found in [13], [20].

Another feature that can be added to this model are tariffs or taxes. In the case where a tax is applied at the supply point, the third general inequality in (2) is replaced by

\[
p^s_i(1 + t_i) + c_{i,j} \geq p^d_j, \forall (i,j) \in \mathcal{A}.
\]

The taxes can be made endogenous to the model, details are found in [35].

The key point is that with either of the above modifications, the complementarity problem is not just the optimality conditions of a linear program. In many cases, there is no optimization problem corresponding to the complementarity conditions.

Nonlinear Complementarity Problem

We now abstract from the particular example to describe more carefully the complementarity problem in its mathematical form. All the above examples can be cast as nonlinear complementarity problems (NCPs) defined as follows:

\[
\text{(NCP) Given a function } F : \mathbb{R}^n \rightarrow \mathbb{R}^n, \text{ find } z \in \mathbb{R}^n \text{ such that}
\]

\[
0 \leq z \perp F(z) \geq 0.
\]

Recall that the \( \perp \) sign signifies that one of the inequalities is satisfied as an equality, so that componentwise, \( z_i F_i(z) = 0 \). We frequently refer to this property as \( z \) is “complementary” to \( F \). A special case of the NCP that has received much attention is when \( F \) is a linear function, the linear complementarity problem [8].
1.2 Walrasian Equilibrium

A Walrasian equilibrium can also be formulated as a complementarity problem (see [33]). In this case, we want to find a price \( p \in \mathbb{R}^m \) and an activity level \( y \in \mathbb{R}^n \) such that

\[
\begin{align*}
0 &\leq y \perp L(p) := -A^T p \geq 0 \\
0 &\leq p \perp S(p,y) := b + Ay - d(p) \geq 0
\end{align*}
\]

where \( S(p,y) \) represents the excess supply function and \( L(p) \) represents the loss function. Complementarity allows us to choose the activities \( y_j \) to run (i.e. only those that do not make a loss). The second set of inequalities state that the price of a commodity can only be positive if there is no excess supply. These conditions indeed correspond to the standard exposition of Walras’ law which states that supply equals demand if we assume all prices \( p \) will be positive at a solution. Formulations of equilibria as systems of equations do not allow the model to choose the activities present, but typically make an a priori assumption on this matter.

GAMS Code

A GAMS implementation of (3) is given in Figure 2. Many large scale models of this nature have been developed. An interested modeler could, for example, see how a large scale complementarity problem was used to quantify the effects of the Uruguay round of talks [26].

Figure 2: Walrasian equilibrium as an NCP, walras1.gms

```
$include walras.dat

positive variables p(i), y(j);

equations S(i), L(j);

S(i).. b(i) + sum(j, A(i,j)*y(j)) - c(i)*sum(k, g(k)*p(k)) / p(i)
     =g= 0;

L(j).. -sum(i, p(i)*A(i,j)) =g= 0;

model walras / S.p, L.y /;
solve walras using mcp;
```

Extension: Intermediate Variables

In many modeling situations, a key tool for clarification is the use of intermediate variables. As an example, the modeler may wish to define a variable corresponding to the demand function \( d(p) \) in the Walrasian equilibrium (3). The syntax for carrying this out is shown in Figure 3 where we use the variables \( d \) to store the demand function referred to in the excess supply equation. The model \( \text{walras} \) now contains a mixture of equations and complementarity constraints. Since constructs of this type are prevalent in many practical models, the GAMS syntax allows such formulations.

Figure 3: Walrasian equilibrium as an MCP, walras2.gms

```
$include walras.dat

positive variables p(i), y(j);
variables d(i);

equations S(i), L(j), demand(i);

demand(i)
  d(i) =e= c(i)*sum(k, g(k)*p(k)) / p(i) ;

S(i).. b(i) + sum(j, A(i,j)*y(j)) - d(i) =g= 0 ;
```
L(j).. -sum(i, p(i)*A(i,j)) =g= 0 ;

model walras / demand.d, S.p, L.y /;
solve walras using mcp;

Note that positive variables are paired with inequalities, while free variables are paired with equations. A crucial point misunderstood by many experienced modelers is that the bounds on the variable determine the relationships satisfied by the function $F$. Thus in Figure 3, $d$ is a free variable and therefore its paired equation demand is an equality. Similarly, since $p$ is nonnegative, its paired relationship $S$ is a (greater-than) inequality.

A simplification is allowed to the model statement in Figure 3. In many cases, it is not significant to match free variables explicitly to equations; we only require that there are the same number of free variables as equations. Thus, in the example of Figure 3, the model statement could be replaced by

```plaintext
model walras / demand, S.p, L.y /;
```

This extension allows existing GAMS models consisting of a square system of nonlinear equations to be easily recast as a complementarity problem - the model statement is unchanged. GAMS generates a list of all variables appearing in the equations found in the model statement, performs explicitly defined pairings and then checks that the number of remaining equations equals the number of remaining free variables. However, if an explicit match is given, the PATH solver can frequently exploit the information for better solution. Note that all variables that are not free and all inequalities must be explicitly matched.

**Mixed Complementarity Problem**

A mixed complementarity problem (MCP) is specified by three pieces of data, namely the lower bounds $\ell$, the upper bounds $u$ and the function $F$.

$$(\text{MCP})$$

Given lower bounds $\ell \in \{\mathbb{R} \cup \{-\infty\}\}^n$, upper bounds $u \in \{\mathbb{R} \cup \{\infty\}\}^n$ and a function $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$, find $z \in \mathbb{R}^n$ such that precisely one of the following holds for each $i \in \{1, \ldots, n\}$:

- $F_i(z) = 0$ and $\ell_i \leq z_i \leq u_i$
- $F_i(z) > 0$ and $z_i = \ell_i$
- $F_i(z) < 0$ and $z_i = u_i$.

These relationships define a general MCP (sometimes termed a rectangular variational inequality). We will write these conditions compactly as

$$\ell \preceq x \preceq u \perp F(x).$$

Note that the nonlinear complementarity problem of Nonlinear Complementarity Problem is a special case of the MCP. For example, to formulate an NCP in the GAMS/MCP format we set

```plaintext
z.lo(I) = 0;
```

or declare

```plaintext
positive variable z;
```

Another special case is a square system of nonlinear equations

$$(\text{NE})$$

Given a function $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ find $z \in \mathbb{R}^n$ such that

$$F(z) = 0.$$
free variable z;

In both the above cases, we must not modify the lower and upper bounds on the variables later (unless we wish to drastically change the problem under consideration).

An advantage of the extended formulation described above is the pairing between "fixed" variables (ones with equal upper and lower bounds) and a component of $F$. If a variable $z_i$ is fixed, then $F_i(z)$ is unrestricted since precisely one of the three conditions in the MCP definition automatically holds when $z_i = \ell_i = u_i$. Thus if a variable is fixed in a GAMS model, the paired equation is completely dropped from the model. This convenient modeling trick can be used to remove particular constraints from a model at generation time. As an example, in economics, fixing a level of production will remove the zero-profit condition for that activity.

Simple bounds on the variables are a convenient modeling tool that translates into efficient mathematical programming tools. For example, specialized codes exist for the bound constrained optimization problem

$$\min f(x) \text{ subject to } \ell \leq x \leq u.$$ 

The first order optimality conditions for this problem class are precisely MCP($\nabla f(x), [\ell, u]$). We can easily see this condition in a one dimensional setting. If we are at an unconstrained stationary point, then $\nabla f(x) = 0$. Otherwise, if $x$ is at its lower bound, then the function must be increasing as $x$ increases, so $\nabla f(x) \geq 0$. Conversely, if $x$ is at its upper bound, then the function must be increasing as $x$ decreases, so that $\nabla f(x) \leq 0$. The MCP allows such problems to be easily and efficiently processed.

Upper bounds can be used to extend the utility of existing models. For example, in Figure 3 it may be necessary to have an upper bound on the activity level $y$. In this case, we simply add an upper bound to $y$ in the model statement, and replace the loss equation with the following definition:

$$y_.up(j) = 10;$$
$$L(j).. -\text{sum}(i, p(i)*A(i,j)) =e= 0 ;$$

Here, for bounded variables, we do not know beforehand if the constraint will be satisfied as an equation, less than inequality or greater than inequality, since this determination depends on the values of the solution variables. We adopt the convention that all bounded variables are paired to equations. Further details on this point are given in Listing File. However, let us interpret the relationships that the above change generates. If $y_j = 0$, the loss function can be positive since we are not producing in the $j$th sector. If $y_j$ is strictly between its bounds, then the loss function must be zero by complementarity; this is the competitive assumption. However, if $y_j$ is at its upper bound, then the loss function can be negative. Of course, if the market does not allow free entry, some firms may operate at a profit (negative loss). For more examples of problems, the interested reader is referred to [10], [17], [18].

### 1.3 Solution

We will assume that a file named transmcp.gms has been created using the GAMS syntax which defines an MCP model transport as developed in Transportation Problem. The modeler has a choice of the complementarity solver to use. We are going to further assume that the modeler wants to use PATH.

There are two ways to ensure that PATH is used as opposed to any other GAMS/MCP solver. These are as follows:

1. Add the following line to the transmcp.gms file prior to the solve statement

   ```gams
   option mcp = path;
   ```

   PATH will then be used instead of the default solver provided.

2. Rerun the gamsinst program from the GAMS system directory and choose PATH as the default solver for MCP.

To solve the problem, the modeler executes the command:

```gams
gams transmcp
```
where `transmcp` can be replaced by any filename containing a GAMS model. Many other command line options for GAMS exist; the reader is referred to [4] for further details.

At this stage, control is handed over to the solver which creates a log providing information on what the solver is doing as time elapses. See Section PATH for details about the log file. After the solver terminates, a listing file is generated containing the solution to the problem. We now describe the output in the listing file specifically related to complementarity problems.

**Listing File**

The listing file is the standard GAMS mechanism for reporting model results. This file contains information regarding the compilation process, the form of the generated equations in the model, and a report from the solver regarding the solution process.

We now detail the last part of this output, an example of which is given in Figure 4. We use "..." to indicate where we have omitted continuing similar output.

**Figure 4:** Listing File for solving `transmcp.gms`

```
SOLVE SUMMARY

MODEL TRANSPORT
TYPE MCP
SOLVER PATH FROM LINE 45

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 1 OPTIMAL

RESOURCE USAGE, LIMIT 0.057 1000.000
ITERATION COUNT, LIMIT 31 10000
EVALUATION ERRORS 0 0

Work space allocated -- 0.06 Mb

---- EQU RATIONAL

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle .new-york</td>
<td>-0.225</td>
<td>-0.225</td>
<td>+INF</td>
</tr>
<tr>
<td>seattle .chicago</td>
<td>-0.153</td>
<td>-0.153</td>
<td>+INF</td>
</tr>
<tr>
<td>seattle .topeka</td>
<td>-0.162</td>
<td>-0.126</td>
<td>+INF</td>
</tr>
</tbody>
</table>

...

---- VAR X shipment quantities in cases

<table>
<thead>
<tr>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle .new-york</td>
<td>.</td>
<td>50.000</td>
<td>+INF</td>
</tr>
<tr>
<td>seattle .chicago</td>
<td>.</td>
<td>300.000</td>
<td>+INF</td>
</tr>
</tbody>
</table>

...

**** REPORT SUMMARY :
0 NONOPT
0 INFEASIBLE
0 UNBOUNDED
0 REDEFINED
```
After a summary line indicating the model name and type and the solver name, the listing file shows a solver status and a model status. Table 1 and Table 2 display the relevant codes that are returned under different circumstances. A modeler can access these codes within the transmcp.gms file using transport.solstat and transport.modelstat respectively.

**Table 1: Solver Status Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal completion</td>
<td>Solver returned to GAMS without an error</td>
</tr>
<tr>
<td>2</td>
<td>Iteration interrupt</td>
<td>Solver used too many iterations</td>
</tr>
<tr>
<td>3</td>
<td>Resource interrupt</td>
<td>Solver took too much time</td>
</tr>
<tr>
<td>4</td>
<td>Terminated by solver</td>
<td>Solver encountered difficulty and was unable to continue</td>
</tr>
<tr>
<td>8</td>
<td>User interrupt</td>
<td>The user interrupted the solution process</td>
</tr>
</tbody>
</table>

**Table 2: Model Status Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Optimal</td>
<td>Solver found a solution of the problem</td>
</tr>
<tr>
<td>6</td>
<td>Intermediate infeasible</td>
<td>Solver failed to solve the problem</td>
</tr>
</tbody>
</table>

After this, a listing of the time and iterations used is given, along with a count on the number of evaluation errors encountered. If the number of evaluation errors is greater than zero, further information can typically be found later in the listing file, prefaced by "∗∗∗∗". Information provided by the solver is then displayed.

Next comes the solution listing, starting with each of the equations in the model. For each equation passed to the solver, four columns are reported, namely the lower bound, level, upper bound and marginal. GAMS moves all parts of a constraint involving variables to the left hand side, and accumulates the constants on the right hand side. The lower and upper bounds correspond to the constants that GAMS generates. For equations, these should be equal, whereas for inequalities one of them should be infinite. The level value of the equation (an evaluation of the left hand side of the constraint at the current point) should be between these bounds, otherwise the solution is infeasible and the equation is marked as follows:

```
seattle .chicago  -0.153  -2.000  +INF  300.000  INFES
```

The marginal column in the equation contains the value of the the variable that was matched with this equation.

For the variable listing, the lower, level and upper columns indicate the lower and upper bounds on the variables and the solution value. The level value returned by PATH will always be between these bounds. The marginal column contains the value of the slack on the equation that was paired with this variable. If a variable appears in one of the constraints in the model statement but is not explicitly paired to a constraint, the slack reported here contains the internally matched constraint slack. The definition of this slack is the minimum of equ.l - equ.lower and equ.l - equ.upper, where equ is the paired equation.

Finally, a summary report is given that indicates how many errors were found. Figure 4 is a good case; when the model has infeasibilities, these can be found by searching for the string "INFES" as described above.

**Redefined Equations**

Unfortunately, this is not the end of the story. Some equations may have the following form:

```
LOWER  LEVEL  UPPER  MARGINAL
new-york  325.000  350.000  325.000  0.225 REDEF
```

This should be construed as a warning from GAMS, as opposed to an error. In principle, the REDEF should only occur if the paired variable to the constraint had a finite lower and upper bound and the variable is at one of those bounds. In this case,
at the solution of the complementarity problem the "equation (=e=)" may not be satisfied. The problem occurs because of a limitation in the GAMS syntax for complementarity problems. The GAMS equations are used to define the function $F$. The bounds on the function $F$ are derived from the bounds on the associated variable. Before solving the problem, for finite bounded variables, we do not know if the associated function will be positive, negative or zero at the solution. Thus, we do not know whether to define the equation as "=e=", "=l=" or "=g=". GAMS therefore allows any of these, and informs the modeler via the "REDEF" label that internally GAMS has redefined the bounds so that the solver processes the correct problem, but that the solution given by the solver does not satisfy the original bounds. However, in practice, a "REDEF" can also occur when the equation is defined using "=e=" and the variable has a single finite bound. This is allowed by GAMS, and as above, at a solution of the complementarity problem, the variable is at its bound and the function $F$ does not satisfy the "=e=" relationship.

*Note that this is not an error, just a warning. The solver has solved the complementarity problem specified by this equation.* GAMS gives this report to ensure that the modeler understands that the complementarity problem derives the relationships on the equations from the bounds, not from the equation definition.

### 1.4 Pitfalls

As indicated above, the ordering of an equation is important in the specification of an MCP. Since the data of the MCP is the function $F$ and the bounds $\ell$ and $u$, it is important for the modeler to pass the solver the function $F$ and not $-F$.

For example, if we have the optimization problem,

$$\min_{x \in [0,2]} (x - 1)^2$$

then the first order optimality conditions are

$$0 \leq x \leq 2 \perp 2(x - 1)$$

which has a unique solution, $x = 1$. Figure 5 provides correct GAMS code for this problem.

**Figure 5:** First order conditions as an MCP, `first.gms`

```gams
variables x;
equations d_f;
x.l = 0;
x.u = 2;
d_f.. 2*(x - 1) =e= 0;
model first / d_f.x /;
solve first using mcp;
```

However, if we accidentally write the valid equation

```
d_f.. 0 =e= 2*(x - 1);
```

the problem given to the solver is

$$0 \leq x \leq 2 \perp -2(x - 1)$$

which has three solutions, $x = 0$, $x = 1$, and $x = 2$. This problem is in fact the stationary conditions for the nonconvex quadratic problem,

$$\max_{x \in [0,2]} (x - 1)^2,$$

not the problem we intended to solve.

Continuing with the example, when $x$ is a free variable, we might conclude that the ordering of the equation is irrelevant because we always have the equation, $2(x - 1) = 0$, which does not change under multiplication by $-1$. In most cases, the
ordering of equations (which are complementary to free variables) does not make a difference since the equation is internally "substituted out" of the model. In particular, for defining equations, such as that presented in Figure 3., the choice appears to be arbitrary.

However, in difficult (singular or near singular) cases, the substitution cannot be performed, and instead a perturbation is applied to $F$, in the hope of "(strongly) convexifying" the problem. If the perturbation happens to be in the wrong direction because $F$ was specified incorrectly, the perturbation actually makes the problem less convex, and hence less likely to solve. Note that determining which of the above orderings of the equations makes most sense is typically tricky. One rule of thumb is to check whether if you replace the "=e=" by "=g="; and then increase "x", is the inequality intuitively more likely to be satisfied. If so, you probably have it the right way round, if not, reorder.

Furthermore, underlying model convexity is important. For example, if we have the linear program

$$\begin{align*}
\min_x & \quad c^T x \\
\text{subject to} & \quad Ax = b, x \geq 0
\end{align*}$$

we can write the first order optimality conditions as either

$$\begin{align*}
0 & \leq x \perp -A^T \mu + c \\
\mu \text{ free} & \perp Ax - b
\end{align*}$$

or, equivalently,

$$\begin{align*}
0 & \leq x \perp -A^T \mu + c \\
\mu \text{ free} & \perp b - Ax
\end{align*}$$

because we have an equation. The former is a linear complementarity problem with a positive semidefinite matrix, while the latter is almost certainly indefinite. Also, if we need to perturb the problem because of numerical problems, the former system will become positive definite, while the later becomes highly nonconvex and unlikely to solve.

Finally, users are strongly encouraged to match equations and free variables when the matching makes sense for their application. Structure and convexity can be destroyed if it is left to the solver to perform the matching. For example, in the above example, we could loose the positive semidefinite matrix with an arbitrary matching of the free variables.

## 2 PATH

Newton’s method, perhaps the most famous solution technique, has been extensively used in practice to solve to square systems of nonlinear equations. The basic idea is to construct a local approximation of the nonlinear equations around a given point, $x^k$, solve the approximation to find the Newton point, $x^N$, update the iterate, $x^{k+1} = x^N$, and repeat until we find a solution to the nonlinear system. This method works extremely well close to a solution, but can fail to make progress when started far from a solution. To guarantee progress is made, a line search between $x^k$ and $x^N$ is used to enforce sufficient decrease on an appropriately defined merit function. Typically, $1/2 \|F(x)\|^2$ is used.

PATH uses a generalization of this method on a nonsmooth reformulation of the complementarity problem. To construct the Newton direction, we use the normal map [34] representation

$$F(\pi(x)) + x - \pi(x)$$

associated with the MCP, where $\pi(x)$ represents the projection of $x$ onto $[\ell, u]$ in the Euclidean norm. We note that if $x$ is a zero of the normal map, then $\pi(x)$ solves the MCP. At each iteration, a linearization of the normal map, a linear complementarity problem, is solved using a pivotal code related to Lemke’s method.

Versions of PATH prior to 4.x are based entirely on this formulation using the residual of the normal map

$$\|F(\pi(x)) + x - \pi(x)\|$$

as a merit function. However, the residual of the normal map is not differentiable, meaning that if a subproblem is not solvable then a "steepest descent" step on this function cannot be taken. PATH 4.x considers an alternative nonsmooth system [21], $\Phi(x) = 0$, where $\Phi_i(x) = \phi(x_i, F_i(x))$ and $\phi(a, b) := \sqrt{a^2 + b^2} - a - b$. The merit function, $\|\Phi(x)\|^2$, in this case is differentiable, and is used for globalization purposes. When the subproblem solver fails, a projected gradient direction for this merit function is searched. It is shown in [19] that this provides descent and a new feasible point to continue PATH, and convergence to stationary points and/or solutions of the MCP is provided under appropriate conditions.
The remainder of this chapter details the interpretation of output from PATH and ways to modify the behavior of the code. To this end, we will assume that the modeler has created a file named `transmcp.gms` which defines an MCP model `transport` as described in Section Transportation Problem and is using PATH 4.x to solve it. See Section Solution for information on changing the solver.

### 2.1 Log File

We will now describe the behavior of the PATH algorithm in terms of the output typically produced. An example of the log for a particular run is given in Figure 6 and Figure 7.

**Figure 6:** Log File from PATH for solving `transmcp.gms`

```
--- Starting compilation
--- trnsmcp.gms(46) 1 Mb
--- Starting execution
--- trnsmcp.gms(27) 1 Mb
--- Generating model transport
--- trnsmcp.gms(45) 1 Mb
--- 11 rows, 11 columns, and 24 non-zeroes.
--- Executing PATH
  Work space allocated -- 0.06 Mb
  Reading the matrix.
  Reading the dictionary.
Path v4.3: GAMS Link ver037, SPARC/SOLARIS
11 row/cols, 35 non-zeros, 28.93% dense.

Path 4.3 (Sat Feb 26 09:38:08 2000)
Written by Todd Munson, Steven Dirkse, and Michael Ferris

INITIAL POINT STATISTICS
Maximum of X . . . . . . . . . . . -0.0000e+00 var: (x.seattle.new-york)
Maximum of F . . . . . . . . . . . 6.0000e+02 eqn: (supply.san-diego)
Maximum of Grad F . . . . . . . . . 1.0000e+00 eqn: (demand.new-york)  
  var: (x.seattle.new-york)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm . . . . . . . . . . 3.0000e+00 eqn: (supply.seattle)
Minimum Row Norm . . . . . . . . . . 2.0000e+00 eqn: (rational.seattle.new-york)
Maximum Column Norm . . . . . . . . . . 3.0000e+00 var: (p_supply.seattle)
Minimum Column Norm . . . . . . . . . . 2.0000e+00 var: (x.seattle.new-york)

Crash Log
major  func  diff  size  residual  step  prox  (label)
  0  0 ieres06  1.0416e+03  0.0e+00  (demand.new-york)
  1  1 3 3 8.3096e+02  1.0e+00  1.0e+01  (demand.new-york)  
pn_search terminated: no basis change.
```

**Figure 7:** Log File from PATH for solving `transmcp.gms` (continued)

```
Major Iteration Log
major  minor  func  grad  residual  step  type  prog  inorm  (label)
  0  0  2  2  1.0029e+03  I  9.0e+00  6.2e+02  (demand.new-york)
  1  1  3  3  8.3096e+02  1.0e+00  SO  3.6e+00  4.5e+02  (demand.new-york)

...
The first few lines on this log file are printed by GAMS during its compilation and generation phases. The model is then passed off to PATH at the stage where the "Executing PATH" line is written out. After some basic memory allocation and problem checking, the PATH solver checks if the modeler required an option file to be read. In the example this is not the case. If PATH is directed to read an option file (see PATH Options below), then the following output is generated after the PATH banner.

The second block provides more information about the Jacobian at the starting point. These can be used to help scale the model. See Section Advanced Topics for complete details.

Crash Log

The first phase of the code is a crash procedure attempting to quickly determine which of the inequalities should be active. This procedure is documented fully in \[12\], and an example of the Crash Log can be seen in Figure 6. The first column of
the crash log is just a label indicating the current iteration number, the second gives an indication of how many function evaluations have been performed so far. Note that precisely one Jacobian (gradient) evaluation is performed per crash iteration. The number of changes to the active set between iterations of the crash procedure is shown under the "diff" column. The crash procedure terminates if this becomes small. Each iteration of this procedure involves a factorization of a matrix whose size is shown in the next column. The residual is a measure of how far the current iterate is from satisfying the complementarity conditions (MCP); it is zero at a solution. See Merit Functions for further information. The column "step" corresponds to the step length taken in this iteration - ideally this should be 1. If the factorization fails, then the matrix is perturbed by an identity matrix scaled by the value indicated in the "prox" column. The "label" column indicates which row in the model is furthest away from satisfying the conditions (MCP). Typically, relatively few crash iterations are performed. Section PATH Options gives mechanisms to affect the behavior of these steps.

**Major Iteration Log**

After the crash is completed, the main algorithm starts as indicated by the "Major Iteration Log" flag (see Figure 7). The columns that have the same labels as in the crash log have precisely the same meaning described above. However, there are some new columns that we now explain. Each major iteration attempts to solve a linear mixed complementarity problem using a pivotal method that is a generalization of Lemke's method [31]. The number of pivots performed per major iteration is given in the "minor" column.

The "grad" column gives the cumulative number of Jacobian evaluations used; typically one evaluation is performed per iteration. The "inorm" column gives the value of the error in satisfying the equation indicated in the "label" column.

At each iteration of the algorithm, several different step types can be taken, due to the use of nonmonotone searches [11], [14] which are used to improve robustness. In order to help the PATH user, we have added two code letters indicating the return code from the linear solver and the step type to the log file. Table 3 explains the return codes for the linear solver and Table 4 explains the meaning of each step type. The ideal output in this column is either "SO", with "SD" and "SB" also being reasonable. Codes different from these are not catastrophic, but typically indicate the solver is having difficulties due to numerical issues or nonconvexities in the model.

**Table 3: Linear Solver Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>A cycle was detected.</td>
</tr>
<tr>
<td>E</td>
<td>An error occurred in the linear solve.</td>
</tr>
<tr>
<td>I</td>
<td>The minor iteration limit was reached.</td>
</tr>
<tr>
<td>N</td>
<td>The basis became singular.</td>
</tr>
<tr>
<td>R</td>
<td>An unbounded ray was encountered.</td>
</tr>
<tr>
<td>S</td>
<td>The linear subproblem was solved.</td>
</tr>
<tr>
<td>T</td>
<td>Failed to remain within tolerance after factorization was performed.</td>
</tr>
</tbody>
</table>

**Table 4: Step Type Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A Backtracking search was performed from the current iterate to the Newton point in order to obtain sufficient decrease in the merit function.</td>
</tr>
<tr>
<td>D</td>
<td>The step was accepted because the Distance between the current iterate and the Newton point was small.</td>
</tr>
<tr>
<td>G</td>
<td>A gradient step was performed.</td>
</tr>
<tr>
<td>I</td>
<td>Initial information concerning the problem is displayed.</td>
</tr>
<tr>
<td>M</td>
<td>The step was accepted because the Merit function value is smaller than the nonmonotone reference value.</td>
</tr>
<tr>
<td>O</td>
<td>A step that satisfies both the distance and merit function tests.</td>
</tr>
<tr>
<td>R</td>
<td>A Restart was carried out.</td>
</tr>
<tr>
<td>W</td>
<td>A Watchdog step was performed in which we returned to the last point encountered with a better merit function value than the nonmonotone reference value (M, O, or B step), regenerated the Newton point, and performed a backtracking search.</td>
</tr>
</tbody>
</table>
Minor Iteration Log

If more than 500 pivots are performed, a minor log is output that gives more details of the status of these pivots. A listing from transmcp model follows, where we have set the output_minor_iteration_frequency option to 1.

<table>
<thead>
<tr>
<th>Minor Iteration Log</th>
<th>minor</th>
<th>t</th>
<th>z</th>
<th>w</th>
<th>v</th>
<th>art</th>
<th>ckpts</th>
<th>enter</th>
<th>leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.2538e-01</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t[0]</td>
<td>z[11]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9.0823e-01</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>w[11]</td>
<td>w[10]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.0000e+00</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>z[10]</td>
<td>t[0]</td>
<td></td>
</tr>
</tbody>
</table>

t is a parameter that goes from zero to 1 for normal starts in the pivotal code. When the parameter reaches 1, we are at a solution to the subproblem. The t column gives the current value for this parameter. The next columns report the current number of problem variables z and slacks corresponding to variables at lower bound w and at upper bound v. Artificial variables are also noted in the minor log, see [15] for further details. Checkpoints are times where the basis matrix is refactorized. The number of checkpoints is indicated in the ckpts column. Finally, the minor iteration log displays the entering and leaving variables during the pivot sequence.

Restart Log

The PATH code attempts to fully utilize the resources provided by the modeler to solve the problem. Versions of PATH after 3.0 have been much more aggressive in determining that a stationary point of the residual function has been encountered. When it is determined that no progress is being made, the problem is restarted from the initial point supplied in the GAMS file with a different set of options. These restarts give the flexibility to change the algorithm in the hopes that the modified algorithm leads to a solution. The ordering and nature of the restarts were determined by empirical evidence based upon tests performed on real-world problems.

The exact options set during the restart are given in the restart log, part of which is reproduced below.

```
Restart Log
proximal_perturbation 0
crash_method none
crash_perturb yes
nms_initial_reference_factor 2
proximal_perturbation 1.0000e-01
```

If a particular problem solves under a restart, a modeler can circumvent the wasted computation by setting the appropriate options as shown in the log. Note that sometimes an option is repeated in this log. In this case, it is the last option that is used.

Solution Log

A solution report is now given by the algorithm for the point returned. The first component is an evaluation of several different merit functions. Next, a display of some statistics concerning the final point is given. This report can be used detect problems with the model and solution as detailed in Section Advanced Topics.

At the end of the log file, summary information regarding the algorithm’s performance is given. The string "** EXIT - solution found." is an indication that PATH solved the problem. Any other EXIT string indicates a termination at a point that may not be a solution. These strings give an indication of what modelstat and solstat will be returned to GAMS. After this, the "Restarting execution" flag indicates that GAMS has been restarted and is processing the results passed back by PATH.
2.2 Status File

If for some reason the PATH solver exits without writing a solution, or the `sysout` flag is turned on, the status file generated by the PATH solver will be reported in the listing file. The status file is similar to the log file, but provides more detailed information. The modeler is typically not interested in this output.

2.3 User Interrupts

A user interrupt can be effected by typing Ctrl-C. We only check for interrupts every major iteration. If a more immediate response is wanted, repeatedly typing Ctrl-C will eventually kill the job. The number needed is controlled by the `interrupt_limit` option. In this latter case, when a kill is performed, no solution is written and an execution error will be generated in GAMS.

2.4 PATHC

PATHC uses a different link to the GAMS system with the remaining code identical. PATHC does not support MPSGE models, but enables the use of preprocessing and can be used to solve constrained systems of nonlinear equations. The output for PATHC is identical to the main distribution described in Section Log File with additional output for preprocessing. The options are the same between the two versions.

2.5 Preprocessing

The preprocessor is work in progress. The exact output in the final version may differ from that given below.

The purpose of a preprocessor is to reduce the size and complexity of a model to achieve improved performance by the main algorithm. Another benefit of the analysis performed is the detection of some provably unsolvable problems. A comprehensive preprocessor has been incorporated into PATHC as developed in [16].

The preprocessor reports its finding with some additional output to the log file. This output occurs before the initial point statistics. An example of the preprocessing on the `forcebsm` model is presented below.

```
Zero:  0  Single:  112  Double:  0  Forced:  0
Preprocessed size:  72
```

The preprocessor looks for special polyhedral structure and eliminates variables using this structure. These are indicated with the above line of text. Other special structure is also detected and reported.

On exit from the algorithm, we must generate a solution for the original problem. This is done during the postsolve. Following the postsolve, the residual using the original model is reported.

```
Postsolved residual: 1.0518e-10
```

This number should be approximately the same as the final residual reported on the presolved model.

Constrained Nonlinear Systems

Modelers typically add bounds to their variables when attempting to solve nonlinear problems in order to restrict the domain of interest. For example, many square nonlinear systems are formulated as

\[ F(z) = 0, \quad \ell \leq z \leq u, \]

where typically, the bounds on \( z \) are inactive at the solution. This is not an MCP, but is an example of a "constrained nonlinear system" (CNS). It is important to note the distinction between MCP and CNS. The MCP uses the bounds to infer relationships on the function \( F \). If a finite bound is active at a solution, the corresponding component of \( F \) is only constrained to be
nonnegative or nonpositive in the MCP, whereas in CNS it must be zero. Thus there may be many solutions of MCP that do
not satisfy $F(z) = 0$. Only if $z^*$ is a solution of MCP with $\ell < z^* < u$ is it guaranteed that $F(z^*) = 0$.

Internally, PATHC reformulates a constrained nonlinear system of equations to an equivalent complementarity problem. The
reformulation adds variables, $y$, with the resulting problem written as:

\[
\ell \leq x \leq u \quad \perp \quad -y \\
y \text{ free} \quad \perp \quad F(x).
\]

This is the MCP model passed on to the PATH solver.

## 3 PATH Options

The default options of PATH should be sufficient for most models; the technique for changing these options are now described.

To change the default options on the model transport, the modeler is required to write a file path.opt in the working
directory and either add a line

```
transport.optfile = 1;
```

before the solve statement in the file transmcp.gms, or use the command-line option

```
gams transmcp optfile=1
```

Unless the modeler has changed the WORKDIR parameter explicitly, the working directory will be the directory containing
the model file.

We give a list of the available options along with their defaults and meaning in the following. Note that only the first three
characters of every word are significant.

### 3.1 General options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>chen_lambda</td>
<td>lambda parameter for Chen-Chen-Kanzow residual</td>
<td>0.8</td>
</tr>
<tr>
<td>convergence_tolerance</td>
<td>stopping criterion</td>
<td>1e-6</td>
</tr>
<tr>
<td>crash_iteration_limit</td>
<td>maximum iterations allowed in crash</td>
<td>50</td>
</tr>
<tr>
<td>crashMerit_function</td>
<td>merit function used in crash method</td>
<td></td>
</tr>
<tr>
<td></td>
<td>normal Use the normal map</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fischer Use the Fischer function</td>
<td></td>
</tr>
<tr>
<td>crash_method</td>
<td>pnewton or none</td>
<td>pnewton</td>
</tr>
<tr>
<td></td>
<td>pnewton Use projected Newton method</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>crash_minimum_dim</td>
<td>minimum problem dimension to perform crash</td>
<td>1</td>
</tr>
<tr>
<td>crash_nbchange_limit</td>
<td>number of changes to the basis allowed</td>
<td>1</td>
</tr>
<tr>
<td>crash_perturb</td>
<td>perturb the problem using pnewton crash</td>
<td>1</td>
</tr>
<tr>
<td>crash_searchtype</td>
<td>search type to use in the crash method</td>
<td>line</td>
</tr>
<tr>
<td></td>
<td>line Use a linesearch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>arc Use an arcsearch</td>
<td></td>
</tr>
<tr>
<td>cumulativeIterationLimit</td>
<td>maximum minor iterations allowed</td>
<td>10000</td>
</tr>
<tr>
<td>gradient_searchtype</td>
<td>search type to use on a gradient step</td>
<td>arc</td>
</tr>
<tr>
<td></td>
<td>line Use a linesearch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>arc Use an arcsearch</td>
<td></td>
</tr>
<tr>
<td>gradient_step_limit</td>
<td>gradient steps allowed before restarting</td>
<td>5</td>
</tr>
</tbody>
</table>
### Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_crash_iterations</td>
<td>output information on crash iterations</td>
<td>1</td>
</tr>
<tr>
<td>output_crash_iterations_frequency</td>
<td>frequency at which crash iteration log is printed</td>
<td>1</td>
</tr>
<tr>
<td>output_errors</td>
<td>output error messages</td>
<td>1</td>
</tr>
<tr>
<td>output_final_degeneracy_statistics</td>
<td>print information regarding degeneracy at the solution</td>
<td>0</td>
</tr>
<tr>
<td>output_final_point</td>
<td>output final point returned from PATH</td>
<td>0</td>
</tr>
<tr>
<td>output_final_point_statistics</td>
<td>output information about the point, function, and Jacobian at the final point</td>
<td>1</td>
</tr>
<tr>
<td>output_final_scaling_statistics</td>
<td>display matrix norms on the Jacobian at the final point</td>
<td>0</td>
</tr>
<tr>
<td>output_final_statistics</td>
<td>output evaluation of available merit functions at the final point</td>
<td>1</td>
</tr>
<tr>
<td>output_final_summary</td>
<td>output summary information</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_point</td>
<td>output initial point given to PATH</td>
<td>0</td>
</tr>
<tr>
<td>output_initial_point_statistics</td>
<td>output information about the point, function, and Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_scaling_statistics</td>
<td>display matrix norms on the Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_statistics</td>
<td>output evaluation of available merit functions at the initial point</td>
<td>0</td>
</tr>
<tr>
<td>output_linear_model</td>
<td>output linear model at each major iteration</td>
<td>0</td>
</tr>
</tbody>
</table>
GAMS controls the total number of pivots allowed via the \texttt{iterlim} option. If more pivots are needed for a particular model then either of the following lines should be added to the \texttt{transmcp.gms} file before the solve statement:

\begin{verbatim}
option iterlim = 2000;
transport.iterlim = 2000;
\end{verbatim}

Similarly if the solver runs out of memory, then the workspace allocated can be changed using

\begin{verbatim}
transport.workspace = 20;
\end{verbatim}

The above example would allocate 20MB of workspace for solving the model.

Problems with a singular basis matrix can be overcome by using the \texttt{proximal_perturbation} option \cite{3}, and linearly dependent columns can be output with the \texttt{output_factorization_singularities} option. For more information on singularities, we refer the reader to Section Advanced Topics.

As a special case, PATH can emulate Lemke’s method \cite{7}, \cite{31} for LCP with the following options:

\begin{verbatim}
crash_method none;
crash_perturb no;
major_iteration_limit 1;
lemke_start first;
nms no;
\end{verbatim}

If instead, PATH is to imitate the Successive Linear Complementarity method (SLCP, often called the Josephy Newton method) \cite{28}, \cite{33}, \cite{32} for MCP with an Armijo style linesearch on the normal map residual, then the options to use are:

\begin{verbatim}
crash_method none;
crash_perturb no;
lemke_start always;
nms_initial_reference_factor 1;
nms_memory size 1;
nms_mstep_frequency 1;
nms_searchtype line;
merit_function normal;
\end{verbatim}

Note that \texttt{nms_memory size 1} and \texttt{nms_initial_reference_factor 1} turn off the nonmonotone linesearch, while \texttt{nms_mstep_frequency 1} turns off watchdoging \cite{5}. \texttt{nms_searchtype line} forces PATH to search the line segment between the initial point and the solution to the linear model, while \texttt{merit_function normal} tell PATH to use the normal map for calculating the residual.
4 Advanced Topics

This chapter discusses some of the difficulties encountered when dealing with complementarity problems. We start off with a very formal definition of a complementarity problem which is used in later sections on merit functions and ill-defined, poorly-scaled, and singular models.

4.1 Formal Definition of MCP

The mixed complementarity problem is defined by a function, \( F : D \rightarrow \mathbb{R}^n \) where \( D \subseteq \mathbb{R}^n \) is the domain of \( F \), and possibly infinite lower and upper bounds, \( \ell \) and \( u \). Let \( C := \{ x \in \mathbb{R}^n \mid \ell \leq x \leq u \} \), a Cartesian product of closed (possibly infinite) intervals. The problem is given as

\[
\text{MCP} : \text{find } x \in C \cap D \text{ s.t. } \langle F(x)y - x \rangle \geq 0, \forall y \in C.
\]

This formulation is a special case of the variational inequality problem defined by \( F \) and a (nonempty, closed, convex) set \( C \). Special choices of \( \ell \) and \( u \) lead to the familiar cases of a system of nonlinear equations

\[
F(x) = 0
\]

(generated by \( \ell \equiv -\infty, u \equiv +\infty \)) and the nonlinear complementarity problem

\[
0 \leq x \perp F(x) \geq 0
\]

(generated using \( \ell \equiv 0, u \equiv +\infty \)).

4.2 Algorithmic Features

We now describe some of the features of the PATH algorithm and the options affecting each.

Merit Functions

A solver for complementarity problems typically employs a merit function to indicate the closeness of the current iterate to the solution set. The merit function is zero at a solution to the original problem and strictly positive otherwise. Numerically, an algorithm terminates when the merit function is approximately equal to zero, thus possibly introducing spurious "solutions". The modeler needs to be able to determine with some reasonable degree of accuracy whether the algorithm terminated at solution or if it simply obtained a point satisfying the desired tolerances that is not close to the solution set. For complementarity problems, we can provide several indicators with different characteristics to help make such a determination. If one of the indicators is not close to zero, then there is some evidence that the algorithm has not found a solution. We note that if all of the indicators are close to zero, we are reasonably sure we have found a solution. However, the modeler has the final responsibility to evaluate the "solution" and check that it makes sense for their application.

For the NCP, a standard merit function is

\[
\|(-x)_+, (-F(x))_+, [(x_i)_+ (F_i(x))_+ i]\|
\]

with the first two terms measuring the infeasibility of the current point and the last term indicating the complementarity error. In this expression, we use \((\cdot)_+\) to represent the Euclidean projection of \(x\) onto the nonnegative orthant, that is \((x)_+ = \max(x, 0)\). For the more general MCP, we can define a similar function:

\[
\left\|x - \pi(x), \left(\frac{x_i - \ell_i}{\|\ell\| + 1}\right)_+, (F_i(x))_+, \left(\frac{u_i - x_i}{\|u\| + 1}\right)_+, (-F_i(x))_+ i\right\|
\]

where \(\pi(x)\) represents the Euclidean projection of \(x\) onto \(C\). We can see that if we have an NCP, the function is exactly the one previously given and for nonlinear systems of equations, this becomes \(\|F(x)\|\).

There are several reformulations of the MCP as systems of nonlinear (nonsmooth) equations for which the corresponding residual is a natural merit function. Some of these are as follows:
• Generalized Minimum Map: $x - \pi(x - F(x))$
• Normal Map: $F(\pi(y)) + y - \pi(y)$
• Fischer Function: $\Phi(x)$, where $\Phi_i(x) := \phi(x_i, F(x_i))$ with
  $$
  \phi(a, b) := \sqrt{a^2 + b^2} - a - b.
  $$

Note that $\phi(a, b) = 0$ if and only if $0 \leq a \perp b \geq 0$. A straightforward extension of $\Phi$ to the MCP format is given for example in [19].

In the context of nonlinear complementarity problems the generalized minimum map corresponds to the classic minimum map $\text{min}(x, F(x))$. Furthermore, for NCPs the minimum map and the Fischer function are both local error bounds and were shown to be equivalent in [36]. Figure 10 in the subsequent section plots all of these merit functions for the ill-defined example discussed therein and highlights the differences between them.

The squared norm of $\Phi$, namely $\Psi(x) := \frac{1}{2} \sum \phi(x_i, F_i)^2$, is continuously differentiable on $\mathbb{R}^n$ provided $F$ itself is. Therefore, the first order optimality conditions for the unconstrained minimization of $\Psi(x)$, namely $\nabla \Psi(x) = 0$ give another indication as to whether the point under consideration is a solution of MCP.

The merit functions and the information PATH provides at the solution can be useful for diagnostic purposes. By default, PATH 4.x returns the best point with respect to the merit function because this iterate likely provides better information to the modeler. As detailed in Section PATH Options, the default merit function in PATH 4.x is the Fischer function. To change this behavior the $\text{merit}$.function option can be used.

**Crashing Method**

The crashing technique [12] is used to quickly identify an active set from the user-supplied starting point. At this time, a proximal perturbation scheme [1], [2] is used to overcome problems with a singular basis matrix. The proximal perturbation is introduced in the crash method, when the matrix factored is determined to be singular. The value of the perturbation is based on the current merit function value.

Even if the crash method is turned off, for example via the option $\text{crash}$.method $\text{none}$, perturbation can be added. This is determined by factoring the matrix that crash would have initially formed. This behavior is extremely useful for introducing a perturbation for singular models. It can be turned off by issuing the option $\text{crash}$.perturb $\text{no}$.

**Nonmonotone Searches**

The first line of defense against convergence to stationary points is the use of a nonmonotone linesearch [23], [24], [14]. In this case we define a reference value, $R^k$ and we use this value in test for sufficient decrease: test:

$$
\Psi(x^k + t_k d^k) \leq R^k + t_k \nabla \Psi(x^k)^T d^k.
$$

Depending upon the choice of the reference value, this allows the merit function to increase from one iteration to the next. This strategy can not only improve convergence, but can also avoid local minimizers by allowing such increases.

We now need to detail our choice of the reference value. We begin by letting $\{M_1, \ldots, M_m\}$ be a finite set of values initialized to $\kappa \Psi(x^0)$, where $\kappa$ is used to determine the initial set of acceptable merit function values. The value of $\kappa$ defaults to 1 in the code and can be modified with the $\text{nms}$.initial_reference_factor option; $\kappa = 1$ indicates that we are not going to allow the merit function to increase beyond its initial value.

Having defined the values of $\{M_1, \ldots, M_m\}$ (where the code by default uses $m = 10$), we can now calculate a reference value. We must be careful when we allow gradient steps in the code. Assuming that $d^k$ is the Newton direction, we define $i_0 = \arg \max M_i$ and $R^k = M_{i_0}$. After the nonmonotone linesearch rule above finds $t_k$, we update the memory so that $M_{i_0} = \Psi(x^k + t_k d^k)$, i.e. we remove an element from the memory having the largest merit function value.

When we decide to use a gradient step, it is beneficial to let $x^k = x^{\text{best}}$ where $x^{\text{best}}$ is the point with the absolute best merit function value encountered so far. We then recalculate $d^k = -\nabla \Psi(x^k)$ using the best point and let $R^k = \Psi(x^k)$. That is to say that we force decrease from the best iterate found whenever a gradient step is performed. After a successful step we set $M_i = \Psi(x^k + t_k d^k)$ for all $i \in [1, \ldots, m]$. This prevents future iterates from returning to the same problem area.
A watchdog strategy [5] is also available for use in the code. The method employed allows steps to be accepted when they are “close” to the current iterate. Nonmonotonic decrease is enforced every $m$ iterations, where $m$ is set by the nms_mstep_frequency option.

### Linear Complementarity Problems

PATH solves a linear complementarity problem each major iteration. Let $M \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$, and $B = [l, u]$ be given. $(\hat{z}, \hat{w}, \hat{v})$ solves the linear mixed complementarity problem defined by $M$, $q$, and $B$ if and only if it satisfies the following constrained system of equations:

\[ Mz - w + v + q = 0 \]  
\[ w^T (z - l) = 0 \]  
\[ v^T (u - z) = 0 \]  
\[ z \in B, w \in \mathbb{R}^n_+, v \in \mathbb{R}^n_+ \]  

where $x + \infty = \infty$ for all $x \in \mathbb{R}$ and $0 \cdot \infty = 0$ by convention. A triple, $(\hat{z}, \hat{w}, \hat{v})$, satisfying equations (4) - (6) is called a complementary triple.

The objective of the linear model solver is to construct a path from a given complementary triple $(\hat{z}, \hat{w}, \hat{v})$ to a solution $(\bar{z}, \bar{w}, \bar{v})$. The algorithm used to solve the linear problem is identical to that given in [9]; however, artificial variables are incorporated into the model. The augmented system is then:

\[ Mz - w + v + Da + \frac{(1 - t)}{s} (sr) + q = 0 \]  
\[ w^T (z - l) = 0 \]  
\[ v^T (u - z) = 0 \]  
\[ z \in B, w \in \mathbb{R}^n_+, v \in \mathbb{R}^n_+, a \equiv 0, t \in [0, 1] \]  

where $r$ is the residual, $t$ is the path parameter, and $a$ is a vector of artificial variables. The residual is scaled by $s$ to improve numerical stability.

The addition of artificial variables enables us to construct an initial invertible basis consistent with the given starting point even under rank deficiency. The procedure consists of two parts: constructing an initial guess as to the basis and then recovering from rank deficiency to obtain an invertible basis. The crash technique gives a good approximation to the active set. The first phase of the algorithm uses this information to construct a basis by partitioning the variables into three sets:

1. $W = \{ i \in \{1, \ldots, n\} \mid \hat{z}_i = l_i \text{ and } \hat{w}_i > 0 \}$
2. $V = \{ i \in \{1, \ldots, n\} \mid \hat{z}_i = u_i \text{ and } \hat{w}_i > 0 \}$
3. $Z = \{1, \ldots, n\} \setminus W \cup V$

Since $(\hat{z}, \hat{w}, \hat{v})$ is a complementary triple, $Z \cap W \cap V = \emptyset$ and $Z \cup W \cup V = \{1, \ldots, n\}$. Using the above guess, we can recover an invertible basis consistent with the starting point by defining $D$ appropriately. The technique relies upon the factorization to tell the linearly dependent rows and columns of the basis matrix. Some of the variables may be nonbasic, but not at their bounds. For such variables, the corresponding artificial will be basic.

We use a modified version of EXPAND [22] to perform the ratio test. Variables are prioritized as follows:
1. $t$ leaving at its upper bound.

2. Any artificial variable.

3. Any $z$, $w$, or $v$ variable.

If a choice as to the leaving variable can be made while maintaining numerical stability and sparsity, we choose the variable with the highest priority (lowest number above).

When an artificial variable leaves the basis and a $z$-type variable enters, we have the choice of either increasing or decreasing that entering variable because it is nonbasic but not at a bound. The determination is made such that $t$ increases and stability is preserved.

If the code is forced to use a ray start at each iteration (lemke_start always), then the code carries out Lemke’s method, which is known \[7\] not to cycle. However, by default, we use a regular start to guarantee that the generated path emanates from the current iterate. Under appropriate conditions, this guarantees a decrease in the nonlinear residual. However, it is then possible for the pivot sequence in the linear model to cycle. To prevent this undesirable outcome, we attempt to detect the formation of a cycle with the heuristic that if a variable enters the basis more than a given number of times, we are cycling. The number of times the variable has entered is reset whenever $t$ increases beyond its previous maximum or an artificial variable leaves the basis. If cycling is detected, we terminate the linear solver at the largest value of $t$ and return this point.

Another heuristic is added when the linear code terminates on a ray. The returned point in this case is not the base of the ray. We move a slight distance up the ray and return this new point. If we fail to solve the linear subproblem five times in a row, a Lemke ray start will be performed in an attempt to solve the linear subproblem. Computational experience has shown this to be an effective heuristic and generally results in solving the linear model. Using a Lemke ray start is not the default mode, since typically many more pivots are required.

For time when a Lemke start is actually used in the code, an advanced ray can be used. We basically choose the “closest” extreme point of the polytope and choose a ray in the interior of the normal cone at this point. This helps to reduce the number of pivots required. However, this can fail when the basis corresponding to the cell is not invertible. We then revert to the Lemke start.

Since the EXPAND pivot rules are used, some of the variable may be nonbasic, but slightly infeasible, as the solution. Whenever the linear code finisher, the nonbasic variables are put at their bounds and the basic variable are recomputed using the current factorization. This procedure helps to find the best possible solution to the linear system.

The resulting linear solver as modified above is robust and has the desired property that we start from $({\hat{z}}, {\hat{w}}, {\hat{v}})$ and construct a path to a solution.

**Other Features**

Some other heuristics are incorporated into the code. During the first iteration, if the linear solver fails to find a Newton point, a Lemke start is used. Furthermore, under repeated failures during the linear solve, a Lemke starts will be attempted. A gradient step can also be used when we fail repeatedly.

The proximal perturbation is shrunk each major iteration. However, when numerical difficulties are encountered, it will be increase to a fraction of the current merit function value. These are determined as when the linear solver returns the Reset or Singular status.

Spacer steps are taken every major iteration, in which the iterate is chosen to be the best point for the normal map. The corresponding basis passed into the Lemke code is also updated.

Scaling is done based on the diagonal of the matrix passed into the linear solver.

We finally note, that we the merit function fails to show sufficient decrease over the last 100 iterates, a restart will be performed, as this indicates we are close to a stationary point.
4.3 Difficult Models

Ill-Defined Models

A problem can be ill-defined for several different reasons. We concentrate on the following particular cases. We will call $F$ well-defined at $\bar{x} \in C$ if $\bar{x} \in D$ and ill-defined at $\bar{x}$ otherwise. Furthermore, we define $F$ to be well-defined near $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $C \cap \mathcal{N}(\bar{x}) \subseteq D$. By saying the function is well-defined near $\bar{x}$, we are simply stating that $F$ is defined for all $x \in C$ sufficiently close to $\bar{x}$. A function not well-defined near $\bar{x}$ is termed ill-defined near $\bar{x}$.

We will say that $F$ has a well-defined Jacobian at $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $\mathcal{N}(\bar{x}) \subseteq D$ and $F$ is continuously differentiable on $\mathcal{N}(\bar{x})$. Otherwise the function has an ill-defined Jacobian at $\bar{x}$. We note that a well-defined Jacobian at $\bar{x}$ implies that the MCP has a well-defined function near $\bar{x}$, but the converse is not true.

PATH uses both function and Jacobian information in its attempt to solve the MCP. Therefore, both of these definitions are relevant. We discuss cases where the function and Jacobian are ill-defined in the next two subsections. We illustrate uses for the merit function information and final point statistics within the context of these problems.

**Function Undefined** We begin with a one-dimensional problem for which $F$ is ill-defined at $x = 0$ as follows:

$$0 \leq x \perp \frac{1}{x} \geq 0.$$

Here $x$ must be strictly positive because $\frac{1}{x}$ is undefined at $x = 0$. This condition implies that $F(x)$ must be equal to zero. Since $F(x)$ is strictly positive for all $x$ strictly positive, this problem has no solution.

We are able to perform this analysis because the dimension of the problem is small. Preprocessing linear problems can be done by the solver in an attempt to detect obviously inconsistent problems, reduce problem size, and identify active components at the solution. Similar processing can be done for nonlinear models, but the analysis becomes more difficult to perform. Currently, PATH only checks the consistency of the bounds and removes fixed variables and the corresponding complementary equations from the model.

A modeler might not know a priori that a problem has no solution and might attempt to formulate and solve it. GAMS code for this model is provided in Figure 8. We must specify an initial value for $x$ in the code. If we were to not provide one, GAMS would use $x = 0$ as the default value, notice that $F$ is undefined at the initial point, and terminate before giving the problem to PATH. The error message problem indicates that the function $\frac{1}{x}$ is ill-defined at $x = 0$, but does not determine whether the corresponding MCP problem has a solution.

Figure 8: GAMS Code for Ill-Defined Function

```gams
positive variable x;

equations F;

F.. 1 / x =g= 0;

model simple / F.x /;

x.l = 1e-6;

solve simple using mcp;
```

After setting the starting point, GAMS generates the model, and PATH proceeds to “solve” it. A portion of the output relating to this problem is given in Figure 9 PATH uses the Fischer Function indicator as its termination criteria by default, but evaluates all of the merit functions given in Section Merit Functions at the final point. The Normal Map merit function, and to a lesser extent, the complementarity error, indicate that the “solution” found does not necessarily solve the MCP.

Figure 9: PATH Output for Ill-Defined Function

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inf-Norm of Complementarity</td>
<td>1.0000e+00</td>
<td>(F)</td>
</tr>
<tr>
<td>Inf-Norm of Normal Map</td>
<td>1.1181e+16</td>
<td>(F)</td>
</tr>
<tr>
<td>Inf-Norm of Minimum Map</td>
<td>8.9441e-17</td>
<td>(F)</td>
</tr>
</tbody>
</table>
To indicate the difference between the merit functions, Figure 10 plots them all for the simple example. We note that as $x$ approaches positive infinity, numerically, we are at a solution to the problem with respect to all of the merit functions except for the complementarity error, which remains equal to one. As $x$ approaches zero, the merit functions diverge, also indicating that $x = 0$ is not a solution.

The natural residual and Fischer function tend toward 0 as $x \downarrow 0$. From these measures, we might think $x = 0$ is the solution. However, as previously remarked $F$ is ill-defined at $x = 0$. $F$ and $\nabla F$ become very large, indicating that the function (and Jacobian) might not be well-defined. We might be tempted to conclude that if one of the merit function indicators is not close to zero, then we have not found a solution. This conclusion is not always the case. When one of the indicators is non-zero, we have reservations about the solution, but we cannot eliminate the possibility that we are actually close to a solution. If we slightly perturb the original problem to

$$0 \leq x \perp \frac{1}{x+\varepsilon} \geq 0$$

for a fixed $\varepsilon > 0$, the function is well-defined over $C = \mathbb{R}_n^+$ and has a unique solution at $x = 0$. In this case, by starting at $x > 0$ and sufficiently small, all of the merit functions, with the exception of the Normal Map, indicate that we have solved the problem as is shown by the output in Figure 11 for $\varepsilon = 1 \times 10^{-6}$ and $x = 1 \times 10^{-20}$.

Figure 11: PATH Output for Well-Defined Function
FINAL STATISTICS
Inf-Norm of Complementarity . . 1.0000e-14 eqn: (G)
Inf-Norm of Normal Map. . . . . 1.0000e+06 eqn: (G)
Inf-Norm of Minimum Map . . . . 1.0000e-20 eqn: (G)
Inf-Norm of Fischer Function. . 1.0000e-20 eqn: (G)
Inf-Norm of Grad Fischer Fcn. . 1.0000e-20 eqn: (G)

FINAL POINT STATISTICS
Maximum of X. . . . . . . . . . 1.0000e-20 var: (X)
Maximum of F. . . . . . . . . . 1.0000e+06 eqn: (G)
Maximum of Grad F . . . . . . . 1.0000e+12 var: (X)

In this case, the Normal Map is quite large and we might think that the function and Jacobian are undefined. When only the normal map is non-zero, we may have just mis-identified the optimal basis. By setting the merit function normal option, we can resolve the problem, identify the correct basis, and solve the problem with all indicators being close to zero. This example illustrates the point that all of these tests are not infallible. The modeler still needs to do some detective work to determine if they have found a solution or if the algorithm is converging to a point where the function is ill-defined.

Jacobian Undefined Since PATH uses a Newton-like method to solve the problem, it also needs the Jacobian of $F$ to be well-defined. One model for which the function is well-defined over $C$, but for which the Jacobian is undefined at the solution is: $0 \leq x^\perp - \sqrt{x} \geq 0$. This model has a unique solution at $x = 0$.

Using PATH and starting from the point $x = 1 \times 10^{-14}$, PATH generates the output given in Figure 12.

Figure 12: PATH Output for Ill-Defined Jacobian

FINAL STATISTICS
Inf-Norm of Complementarity . . 1.0000e-07 eqn: (F)
Inf-Norm of Normal Map. . . . . 1.0000e-07 eqn: (F)
Inf-Norm of Minimum Map . . . . 1.0000e-07 eqn: (F)
Inf-Norm of Fischer Function. . 2.0000e-07 eqn: (F)
Inf-Norm of Grad FB Function. . 2.0000e+00 eqn: (F)

FINAL POINT STATISTICS
Maximum of X. . . . . . . . . . 1.0000e-14 var: (X)
Maximum of F. . . . . . . . . . 1.0000e-07 eqn: (F)
Maximum of Grad F . . . . . . . 5.0000e+06 var: (X)

We can see that the gradient of the Fischer Function is nonzero and the Jacobian is beginning to become large. These conditions indicate that the Jacobian is undefined at the solution. It is therefore important for a modeler to inspect the given output to guard against such problems.

If we start from $x = 0$, PATH correctly informs us that we are at the solution. Even though the entries in the Jacobian are undefined at this point, the GAMS interpreter incorrectly returns a value of 0 to PATH. This problem with the Jacobian is therefore undetectable by PATH. (This problem has been fixed in versions of GAMS beyond 19.1).

Poorly Scaled Models

Problems which are well-defined can have various numerical problems that can impede the algorithm’s convergence. One particular problem is a badly scaled Jacobian. In such cases, we can obtain a poor “Newton” direction because of numerical problems introduced in the linear algebra performed. This problem can also lead the code to a point from which it cannot recover.

The final model given to the solver should be scaled such that we avoid numerical difficulties in the linear algebra. The output provided by PATH can be used to iteratively refine the model so that we eventually end up with a well-scaled problem. We note that we only calculate our scaling statistics at the starting point provided. For nonlinear problems these statistics may
not be indicative of the overall scaling of the model. Model specific knowledge is very important when we have a nonlinear problem because it can be used to appropriately scale the model to achieve a desired result.

We look at the `titan.gms` model in MCPLIB, that has some scaling problems. The relevant output from PATH for the original code is given in Figure 13.

**Figure 13:** PATH Output - Poorly Scaled Model

**INITIAL POINT STATISTICS**
- Maximum of X: \(4.1279 \times 10^6\) var: (w.29)
- Maximum of F: \(2.2516 \times 10^0\) eqn: (a1.33)
- Maximum of Grad F: \(6.7753 \times 10^6\) eqn: (a1.29) var: (x1.29)

**INITIAL JACOBIAN NORM STATISTICS**
- Maximum Row Norm: \(9.4504 \times 10^6\) eqn: (a2.29)
- Minimum Row Norm: \(2.7680 \times 10^0\) eqn: (g.10)
- Maximum Column Norm: \(9.4504 \times 10^6\) var: (x1.29)
- Minimum Column Norm: \(1.3840 \times 10^{-3}\) var: (w.10)

The maximum row norm is defined as
\[
\max_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |(\nabla F(x))_{ij}|
\]
and the minimum row norm is
\[
\min_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |(\nabla F(x))_{ij}|.
\]

Similar definitions are used for the column norm. The norm numbers for this particular example are not extremely large, but we can nevertheless improve the scaling. We first decided to reduce the magnitude of the a2 block of equations as indicated by PATH. Using the GAMS modeling language, we can scale particular equations and variables using the .scale attribute. To turn the scaling on for the model we use the .scaleopt model attribute. After scaling the a2 block, we re-ran PATH and found additional blocks of equations that also needed scaling, a2. We also scaled some of the variables, g and w. The code added to the model follows:

```gams
titan.scaleopt = 1;
a1.scale(i) = 1000;
a2.scale(i) = 1000;
g.scale(i) = 1/1000;
w.scale(i) = 100000;
```

After scaling these blocks of equations in the model, we have improved the scaling statistics which are given in Figure 14 for the new model.

**Figure 14:** PATH Output - Well-Scaled Model

**INITIAL POINT STATISTICS**
- Maximum of X: \(1.0750 \times 10^3\) var: (x1.49)
- Maximum of F: \(3.9829 \times 10^{-1}\) eqn: (g.10)
- Maximum of Grad F: \(6.7753 \times 10^3\) eqn: (a1.29) var: (x1.29)

**INITIAL JACOBIAN NORM STATISTICS**
- Maximum Row Norm: \(9.4524 \times 10^3\) eqn: (a2.29)
- Minimum Row Norm: \(2.7680 \times 10^0\) eqn: (g.10)
- Maximum Column Norm: \(9.4904 \times 10^3\) var: (x2.29)
- Minimum Column Norm: \(1.3840 \times 10^{-1}\) var: (w.10)

For this particular problem PATH cannot solve the unscaled model, while it can find a solution to the scaled model. Using the scaling language features and the information provided by PATH we are able to remove some of the problem’s difficulty and obtain better performance from PATH.
It is possible to get even more information on initial point scaling by inspecting the GAMS listing file. The equation row listing gives the values of all the entries of the Jacobian at the starting point. The row norms generated by PATH give good pointers into this source of information.

Not all of the numerical problems are directly attributable to poorly scaled models. Problems for which the Jacobian of the active constraints is singular or nearly singular can also cause numerical difficulty as illustrated next.

### Singular Models

Assuming that the problem is well-defined and properly scaled, we can still have a Jacobian for which the active constraints are singular or nearly singular (i.e., it is ill-conditioned). When problems are singular or nearly singular, we are also likely to have numerical problems. As a result, the “Newton” direction obtained from the linear problem solver can be very bad. In PATH, we can use proximal perturbation or add artificial variables to attempt to remove the singularity problems from the model. However, it is most often beneficial for solver robustness to remove singularities if possible.

The easiest problems to detect are those for which the Jacobian has zero rows and columns. A simple problem for which we have zero rows and columns is:

\[-2 \leq x \leq 2 \quad \perp \quad -x^2 + 1.

Note that the Jacobian, \(-2x\), is non-singular at all three solutions, but singular at the point \(x = 0\). Output from PATH on this model starting at \(x = 0\) is given in Figure 15.

**Figure 15: PATH Output - Zero Rows and Columns**

```
INITIAL POINT STATISTICS
Zero column of order . . . . . . 0.0000e+00 var: (X)
Zero row of order . . . . . . . 0.0000e+00 eqn: (F)
Total zero columns . . . . . . . 1
Total zero rows . . . . . . . . 1
Maximum of F . . . . . . . . . . 1.0000e+00 eqn: (F)
Maximum of Grad F . . . . . . . 0.0000e+00 eqn: (F)
var: (X)
```

We display in the code the variables and equations for which the row/column in the Jacobian is close to zero. These situations are problematic and for nonlinear problems likely stem from the modeler providing an inappropriate starting point or fixing some variables resulting in some equations becoming constant. We note that the solver may perform well in the presence of zero rows and/or columns, but the modeler should make sure that these are what was intended.

Singularities in the model can also be detected by the linear solver. This in itself is a hard problem and prone to error. For matrices which are poorly scaled, we can incorrectly identify ”linearly dependent” rows because of numerical problems. Setting `output_factorization_singularity yes` in an options file will inform the user which equations the linear solver thinks are linearly dependent. Typically, singularity does not cause a lot of problems and the algorithm can handle the situation appropriately. However, an excessive number of singularities are cause for concern. A further indication of possible singularities at the solution is the lack of quadratic convergence to the solution.

### 5 Case Study: Von Thunen Land Model

We now turn our attention towards using the diagnostic information provided by PATH to improve an actual model. The Von Thunen land model, is a problem renowned in the mathematical programming literature for its computational difficulty. We attempt to understand more carefully the facets of the problem that make it difficult to solve. This will enable to outline and identify these problems and furthermore to extend the model to a more realistic and computationally more tractable form.

#### 5.1 Classical Model

The problem is cast in the Arrow-Debreu framework as an equilibrium problem. The basic model is a closed economy consisting of three economic agents, a landowner, a worker and a porter. There is a central market, around which concentric
regions of land are located. Since the produced goods have to be delivered to the market, this is an example of a spatial price equilibrium. The key variables of the model are the prices of commodities, land, labour and transport. Given these prices, it is assumed that the agents demand certain amounts of the commodities, which are supplied so as to maximize profit in each sector. Walras’ law is then a consequence of the assumed competitive paradigm, namely that supply will equal demand in the equilibrium state.

We now describe the problems that the consumers and the producers face. We first look at consumption and derive a demand function for each of the consumer agents in the economy. Each of these agents has a utility function, that they wish to maximize subject to their budgetary constraints. As is typical in such problems, the utility function is assumed to be Cobb-Douglas

\[ u_a(d) = \prod_c d^{\alpha_{c,a}}, \quad \alpha_{c,a} \geq 0, \sum_c \alpha_{c,a} = 1, \]

where the \( \alpha_{c,a} \) are given parameters dependent only on the agent. For each agent \( a \), the variables \( d_c \) represent quantities of the desired commodities \( c \). In the Von Thunen model, the goods are wheat, rice, corn and barley. The agents endowments determine their budgetary constraint as follows. Given current market prices, an agents wealth is the value of the initial endowment bundle for agent \( a \). A closed form solution, corresponding to demand from agent \( a \) for commodity \( c \) is thus

\[ d_{c,a}(p) := \frac{\alpha_{c,a}(p_c,e_a)}{p_c}. \]

Note that this assumes the prices of the commodities \( p_c \) are positive.

The supply side of the economy is similar. The worker earns a wage \( w_L \) for his labour input. The land is distributed around the market in rings with a rental rate \( w_r \) associated with each ring \( r \) of land. The area of land \( a_r \) in each ring is an increasing function of \( r \). The model assumes that labour and land are substitutable via a constant elasticities of substitution (CES) function.

Consider the production \( x_{c,r} \) of commodity \( c \) in region \( r \). In order to maximize profit (or minimize costs), the labour \( y_L \) and land use \( y_r \) solve

\[ \min w_L y_L + w_r y_r \text{ subject to } \phi_c y_L^\beta c y_r^{1-\beta c} \geq x_{c,r}, y_L, y_r \geq 0, \tag{12} \]

where \( \phi_c \) is a given cost function scale parameter, and \( \beta_c \in [0,1] \) is the share parameter. The technology constraint is precisely the CES function allowing a suitable mix of labour and land use. Again, a closed form solution can be calculated. For example, the demand for labour in order to produce \( x_{c,r} \) of commodity \( c \) in region \( r \) is given by

\[ x_{c,r} \beta_c \left( \frac{w_L}{\phi_c w_L} \right)^{\beta_c} \left( \frac{w_r}{1-\beta_c} \right)^{1-\beta_c} \]

Considering all such demands, this clearly assumes the prices of inputs \( w_L, w_r \) are positive. A key point to note is that input commodity (factor) demands to produce \( x_{c,r} \) can be determined by first solving (12) for unit demand \( x_{c,r} \equiv 1 \) and then multiplying these factor demands by the actual amount desired. Let \( \tilde{y}_L \) and \( \tilde{y}_r \) denote the optimal solutions of (12) with \( x_{c,r} \equiv 1 \). Using this fact, the unit production cost \( \gamma_{c,r} \) for commodity \( c \) in region \( r \) can be calculated as follows:

\[ \gamma_{c,r} = w_L \tilde{y}_L + w_r \tilde{y}_r \]

\[ = \frac{\beta_c}{\phi_c} \left( \frac{w_L}{\phi_c w_L} \right)^{\beta_c} \left( \frac{w_r}{1-\beta_c} \right)^{1-\beta_c} + w_r \left( 1 - \beta_c \right) \frac{\beta_c}{\phi_c w_r} \left( \frac{w_r}{1-\beta_c} \right)^{1-\beta_c} \]

\[ = \frac{1}{\phi_c} \left( \frac{w_L}{\beta_c} \right)^{\beta_c} \left( \frac{w_r}{1-\beta_c} \right)^{1-\beta_c}. \]

Transportation is provided by a porter, earning a wage \( w_p \). If we denote the unit cost for transportation of commodity \( c \) by \( t_c \), then unit transportation cost to market is

\[ T_c(x_p) := t_c d_s w_p, \]
where $d_r$ is the distance of region $r$ to the market. Spatial price equilibrium arises from the consideration:

$$0 \leq x_{c,r} \perp \gamma_{c,r}(w_L, w_r) + T_{c,r}(w_p) \geq p_c.$$  

This is intuitively clear; it states that commodity $c$ will be produced in region $r$ only if the combined cost of production and transportation equals the market price.

The above derivations assumed that the producers and consumers acted as price takers. Walras’ law is now invoked to determine the prices so that markets clear. The resulting complementarity problem is:

$$ \gamma_{c,r} = \frac{1}{\phi_c} \left( \frac{w_L}{\hat{b}_c} \right)^{\hat{b}_c} \left( \frac{w_r}{1 - \hat{b}_c} \right)^{1 - \hat{b}_c} $$  

$$0 \leq x_{c,r} \perp \gamma_{c,r} + T_{c,r}(w_p) \geq p_c $$  

$$0 \leq w_L \perp e_L \geq \sum_{r} x_{c,r} \frac{\hat{b}_c \gamma_{c,r}}{w_L} $$  

$$0 \leq w_r \perp a_r \geq \sum_{c} x_{c,r} (1 - \hat{b}_c) \frac{\gamma_{c,r}}{w_r} $$  

$$0 \leq w_p \perp e_p \geq \sum_{r} \lambda_r d_n x_{c,r} $$  

$$0 \leq p_c \perp \sum_{r} x_{c,r} \geq \frac{\alpha_c p e p w_p + \alpha_c L e L w_L + \alpha_c \lambda \sum_r x_{c}, a_r}{p_c} $$

Note that in (15), (16) and (17), the amounts of labour, land and transport are bounded from above, and hence the prices on these inputs are determined as multipliers (or shadow prices) on the corresponding constraints. The final relationship (18) in the above complementarity problem corresponds to market clearance; prices are nonnegative and can only be positive if supply equals demand. (Some modelers multiply the last inequality throughout by $p_c$. This removes problems where $p_c$ becomes zero, but can also introduce spurious solutions.)

The Arrow-Debreu theory guarantees that the problem is homogeneous in prices; $(x, \lambda, w, \lambda, p)$ is also a solution whenever $(x, w, p)$ solves the above. Typically this singularity in the model is removed by fixing a numeraire, that is fixing a price (for example $w_L = 1$) and dropping the corresponding complementary relationship.

Unfortunately, in this formulation even after fixing a numeraire, some of the variables $p$ and $w$ may go to zero, resulting in an ill-defined problem. In the case of the Von Thunen land model, the rental price of land $w_r$ decreases as the distance to market increases, and for remote rings of land, it becomes zero. A standard modeling fix is to put artificial lower bounds on these variables. Even with this fix, the problem typically remains very hard to solve. More importantly, the homogeneity property of the prices used above to fix a numeraire no longer holds, and the corresponding complementary relationship (which was dropped from the problem) may fail to be satisfied. It therefore matters which numeraire is fixed, and many modelers run into difficulty since in many cases the solution found by a solver is invalid for the originally posed model.

In order to test our diagnostic information, we implemented a version of the above model in GAMS. The model corresponds closely to the MCPLIB model pgvon105.gms except we added more regions to make the problem even more difficult. The model file has been documented more fully, and the data rounded to improve clarity.

Our first trial was to solve the model without fixing a numeraire. In this case, PATH 4.x failed to find a solution. At the starting point, the indicators described in Section III-Defined Models are reasonable, and there are no zero rows/columns in the Jacobian. At the best point found, all indicators are still reasonable. However, the listing file indicates a large number of division by zero problems occurring in (16). We also note that a nonzero proximal perturbation is used in the first iteration of the crash method. This is an indication of singularities. We therefore added an option to output factorization singularities, and singularities appeared in the first iteration. At this point, we decided to fix a numeraire to see if this alleviated the problem.

We chose to fix the labour wage rate to 1. After increasing the iterations allowed to 100,000, PATH 4.x solved the problem. The statistics at the solution are cause for concern. In particular, the gradient of the Fischer function is 7 orders of magnitude.
larger than all the other residuals. Furthermore, the Jacobian is very large at the solution point. Looking further in the listing file, a large number of division by zero problems occur in (16).

To track down the problem further, we added an artificial lower bound on the variables \( w_r \) of \( 10^{-5} \), that would not be active at the aforementioned solution. Resolving gave the same "solution", but resulted in the domain errors disappearing.

Although the problem is solved, there is concern on two fronts. Firstly, the gradient of the Fischer function should go to zero at the solution. Secondly, if a modeler happens to make the artificial lower bounds on the variables a bit larger, then they become active at the solution, and hence the constraint that has been dropped by fixing the price of labour at 1 is violated at this point. Of course, the algorithm is unable to detect this problem, since it is not part of the model that is passed to it, and the corresponding output looks satisfactory.

We are therefore led to the conclusion that the model as postulated is ill-defined. The remainder of this section outlines two possible modeling techniques to overcome the difficulties with ill-defined problems of this type.

### 5.2 Intervention Pricing

The principal difficulty is the fact that the rental prices on land go to zero as proximity to the market decreases, and become zero for sufficiently remote rings. Such a property is unlikely to hold in a practical setting. Typically, a landowner has a minimum rental price (for example, land in fallow increases in value). As outlined above, a fixed lower bound on the rental price violates the well-established homogeneity property. A suggestion postulated by Professor Thomas Rutherford is to allow the landowner to intervene and "purchase-back" his land whenever the rental cost gets smaller than a certain fraction of the labour wage.

The new model adds a (homogeneous in price) constraint

\[
0 \leq i_r \perp w_r \geq 0.0001 \times w_L
\]

and modifies (16) and (18) as follows:

\[
0 \leq w_r \perp a_r - i_r \geq \sum_c x_{c,r}(1 - \beta_c) y_{c,r} \frac{w_r}{w_r}
\]

\[
0 \leq p_c \perp \sum_r x_{c,r} \geq \frac{\alpha_{c,p} p w_p + \alpha_{c,d} e_L w_L + \alpha_{c,0} \sum_r w_r (a_r - i_r)}{p_c}.
\]

(19)

Given the intervention purchase, we can now add a lower bound on \( w_r \) to avoid division by zero errors. In our model we chose \( 10^{-5} \) since this will never be active at the solution and therefore will not affect the positive homogeneity. After this reformulation, PATH 4.x solves the problem. Furthermore, the gradient of the Fischer function, although slightly larger than the other residuals, is quite small, and can be made even smaller by reducing the convergence tolerance of PATH. Inspecting the listing file, the only difficulties mentioned are division by zero errors in the market clearance condition (19), that can be avoided a posteori by imposing an artificial (inactive) lower bound on these prices. We chose not to do this however.

### 5.3 Nested Production and Maintenance

Another observation that can be used to overcome the land price going to zero is the fact that land typically requires some maintenance labour input to keep it usable for crop growth. Traditionally, in economics, this is carried out by providing a nested CES function as technology input to the model. The idea is that commodity \( c \) in region \( r \) is made from labour and an intermediate good, which is "maintained land". Essentially, the following production problem replaces (12):

\[
\min_{y_M, y_L, y_r, g} w_L (y_M + y_L) + w_r y_r
\]

subject to

\[
y_r \geq (1 - \beta_c - \varepsilon) g
y_M \geq \varepsilon g
\phi_y y_L g^{1 - \beta} \geq 1,
y_M, y_L, y_r, g \geq 0.
\]

Note that the variable \( y_M \) represents "maintenance labour" and \( g \) represents the amount of "maintained land" produced, an intermediate good. The process of generating maintained land uses a Leontieff production function, namely

\[
\min(\lambda_r y_r, \lambda_M y_M) \geq g.
\]
Here $\lambda_M = \frac{1}{r} \epsilon$ small, corresponds to small amounts of maintenance labour, while $\lambda_r = \frac{1}{1-\beta_c - \epsilon}$ is chosen to calibrate the model correctly. A simple calculus exercise then generates appropriate demand and cost expressions. The resulting complementarity problem comprises (14), (17), (18)

\[
\gamma_{c,r} = \frac{w_L}{\phi_c} \left( \frac{w_L + w_r(1 - \beta_c - \epsilon)}{1 - \beta_c} \right)^{1 - \beta_c}
\]

\[
0 \leq w_L \perp e_L \geq \sum_{c,r} x_{c,r} \gamma_{c,r} \left( \frac{\beta_c + \epsilon(1 - \beta_c)}{w_L + w_r(1 - \beta_c - \epsilon)} \right)
\]

\[
0 \leq w_r \perp a_r \geq \sum_{c} x_{c,r} \gamma_{c,r} (1 - \beta_c)(1 - \beta_c - \epsilon)
\]

After making the appropriate modifications to the model file, PATH 4.x solved the problem on defaults without any difficulties. All indicators showed the problem and solution found to be well-posed.

**Bibliography**


1 Introduction

This document describes the GAMS/PATHNLP solver for non-linear programs and the options unique to this solver.

PATHNLP solves an NLP by internally constructing the Karush-Kuhn-Tucker (KKT) system of first-order optimality conditions associated with the NLP and solving this system using the PATH solver for complementarity problems. The solution to the original NLP is extracted from the KKT solution and returned to GAMS. All of this takes place automatically - no special syntax or user reformulation is required.

Typically, PATHNLP works very well for convex models. It also has a comparative advantage on models whose solution via reduced gradient methods results in a large number of superbasic variables, since the PATH solver won’t construct a dense reduced Hessian in the space of the superbasic variables as reduced gradient solvers do. For nonconvex models, however, PATHNLP is not as robust as the reduced gradient methods.

The theory relating NLP to their KKT systems is well-known: assuming differentiability without convexity, and assuming a constraint qualification holds, then a solution to the NLP must also be a solution to the KKT system. If we also assume convexity, then a solution to the KKT system is also a solution to the NLP - no further constraint qualification is required.

In case PATH fails to find a solution to the KKT system for the NLP, a phase I / phase II method is used in which the phase I objective is simply the feasibility error and the original objective is ignored. If a feasible point is found in phase I then phase II, an attempt to solve the KKT system for the NLP using the current feasible point, is entered.

PATHNLP is installed automatically with your GAMS system. Without a license, it will run in student or demonstration mode (i.e. it will solve small models only). If your GAMS license includes PATH, this size restriction is removed.

2 Usage

If you have installed the system and configured PATHNLP as the default NLP solver, all NLP models without a specific solver option will be solved with PATHNLP. If you installed another solver as the default, you can explicitly request that a particular model be solved using PATHNLP by inserting the statement

```plaintext
option NLP = pathnlp;
```

somewhere before the `solve` statement. Similar comments hold for the other model types (LP, RMINLP, QCP, etc.) PATHNLP can handle.
The standard GAMS model options *iterlim*, *reslim* and *optfile* can be used to control PATHNLP. A description of these options can be found in Section **GAMS Options**, Chapter "Basic Solver Usage". In general this is enough knowledge to solve your models. In some cases, however, you may want to use some of the PATHNLP options to gain further performance improvements or for other reasons. The rules for using an option file are described in in Section **GAMS Options**, in Chapter "Basic Solver Usage". The options used to control PATH can also be used to control PATHNLP. There are also some options unique to PATHNLP described below.

### 3 Options

The table that follows contains the options unique to PATHNLP. For details on the options PATHNLP shares with the other PATH links, see the chapter on the PATH solver.

#### 3.1 General options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>chen_lambda</td>
<td>lambda parameter for Chen-Chen-Kanzow residual</td>
<td>0.8</td>
</tr>
<tr>
<td>convergence_tolerance</td>
<td>stopping criterion</td>
<td>1e-6</td>
</tr>
<tr>
<td>crash_iteration_limit</td>
<td>maximum iterations allowed in crash</td>
<td>50</td>
</tr>
<tr>
<td>crashMeritFunction</td>
<td>merit function used in crash method</td>
<td>fischer</td>
</tr>
<tr>
<td>crashMethod</td>
<td>pnewton or none</td>
<td>pnewton</td>
</tr>
<tr>
<td>crashMinimumDimension</td>
<td>minimum problem dimension to perform crash</td>
<td>1</td>
</tr>
<tr>
<td>crashNbChangeLimit</td>
<td>number of changes to the basis allowed</td>
<td>1</td>
</tr>
<tr>
<td>crashPerturb</td>
<td>perturb the problem using pnewton crash</td>
<td>1</td>
</tr>
<tr>
<td>crashSearchType</td>
<td>search type to use in the crash method</td>
<td>line</td>
</tr>
<tr>
<td>cumulativeIterationLimit</td>
<td>maximum minor iterations allowed</td>
<td>10000</td>
</tr>
<tr>
<td>gradientSearchType</td>
<td>search type to use on a gradient step</td>
<td>arc</td>
</tr>
<tr>
<td>gradientStepLimit</td>
<td>gradient steps allowed before restarting</td>
<td>5</td>
</tr>
<tr>
<td>interruptLimit</td>
<td>ctrl-C’s required before killing job</td>
<td>5</td>
</tr>
<tr>
<td>majorIterationLimit</td>
<td>maximum major iterations allowed</td>
<td>500</td>
</tr>
<tr>
<td>meritFunction</td>
<td>merit function to use (normal or fischer)</td>
<td>fischer</td>
</tr>
<tr>
<td>minorIterationLimit</td>
<td>minor iterations allowed in each major iteration</td>
<td>1000</td>
</tr>
<tr>
<td>nms</td>
<td>allow line searching, watch-dogging, and nonmonotone descent</td>
<td>1</td>
</tr>
<tr>
<td>nmsInitialReferenceFactor</td>
<td>controls size of initial reference value</td>
<td>20</td>
</tr>
<tr>
<td>nmsMaximumWatchdogs</td>
<td>maximum number of watchdog steps allowed</td>
<td>5</td>
</tr>
<tr>
<td>nmsMemorySize</td>
<td>number of reference values kept</td>
<td>10</td>
</tr>
<tr>
<td>nmsMstepFrequency</td>
<td>frequency at which m-steps are performed</td>
<td>10</td>
</tr>
<tr>
<td>nmsSearchType</td>
<td>search type to use</td>
<td>line</td>
</tr>
</tbody>
</table>

For more information on each option, consult the documentation provided with the PATH solver.
3.2 NLP-specific options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow_reform</td>
<td>substitute out objective var and equ when possible</td>
<td>1</td>
</tr>
<tr>
<td>gmo_hess_factor</td>
<td>maximum multiples of Jacobian size to allow Hessian storage: 0=no limit</td>
<td>0</td>
</tr>
<tr>
<td>nlp_lambda</td>
<td>linesearch factor when using the NLP objective</td>
<td>0</td>
</tr>
<tr>
<td>nlp_objective</td>
<td>treat NLP objective differently in PATH linesearch</td>
<td>0</td>
</tr>
<tr>
<td>output_memory</td>
<td>output breakdown of where memory is used</td>
<td>0</td>
</tr>
<tr>
<td>skip_kkt</td>
<td>go right to Phase I / Phase II method</td>
<td>0</td>
</tr>
</tbody>
</table>

3.3 Output options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_crash_iterations</td>
<td>output information on crash iterations</td>
<td>1</td>
</tr>
<tr>
<td>output_crash_iterations_frequency</td>
<td>frequency at which crash iteration log is printed</td>
<td>1</td>
</tr>
<tr>
<td>output_errors</td>
<td>output error messages</td>
<td>1</td>
</tr>
<tr>
<td>output_final_degeneracy_statistics</td>
<td>print information regarding degeneracy at the solution</td>
<td>0</td>
</tr>
<tr>
<td>output_final_point</td>
<td>output final point returned from PATH</td>
<td>0</td>
</tr>
<tr>
<td>output_final_point_statistics</td>
<td>output information about the point, function, and Jacobian at the final point</td>
<td>1</td>
</tr>
<tr>
<td>output_final_scaling_statistics</td>
<td>display matrix norms on the Jacobian at the final point</td>
<td>0</td>
</tr>
<tr>
<td>output_final_statistics</td>
<td>output evaluation of available merit functions at the final point</td>
<td>1</td>
</tr>
<tr>
<td>output_final_summary</td>
<td>output summary information</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_point</td>
<td>output initial point given to PATH</td>
<td>0</td>
</tr>
<tr>
<td>output_initial_point_statistics</td>
<td>output information about the point, function, and Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_scaling_statistics</td>
<td>display matrix norms on the Jacobian at the initial point</td>
<td>1</td>
</tr>
<tr>
<td>output_initial_statistics</td>
<td>output evaluation of available merit functions at the initial point</td>
<td>0</td>
</tr>
<tr>
<td>output_linear_model</td>
<td>output linear model at each major iteration</td>
<td>0</td>
</tr>
<tr>
<td>output_major_iterations</td>
<td>output information on major iterations</td>
<td>1</td>
</tr>
<tr>
<td>output_major_iterations_frequency</td>
<td>frequency at which major iteration log is printed</td>
<td>1</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>output_maximum_zero_listing</td>
<td>limits zero columns reported to listing file</td>
<td>1000</td>
</tr>
<tr>
<td>output_maximum_zero_log</td>
<td>limits zero columns reported to log file</td>
<td>10</td>
</tr>
<tr>
<td>output_minor_iterations</td>
<td>output information on minor iterations</td>
<td>1</td>
</tr>
<tr>
<td>output_minor_iterations_frequency</td>
<td>frequency at which minor iteration log is printed</td>
<td>500</td>
</tr>
<tr>
<td>output_options</td>
<td>output all options and their values</td>
<td>0</td>
</tr>
<tr>
<td>output</td>
<td>no turns all output off</td>
<td>0</td>
</tr>
<tr>
<td>output_preprocess_level</td>
<td>control output of preprocessing information</td>
<td>1</td>
</tr>
<tr>
<td>output_restart_log</td>
<td>output options during restarts</td>
<td>1</td>
</tr>
<tr>
<td>output_time</td>
<td>output breakdown of where time is spent</td>
<td>0</td>
</tr>
<tr>
<td>output_warnings</td>
<td>output warning messages</td>
<td>0</td>
</tr>
</tbody>
</table>
1 Release Notes

- April 30, 2002: Level 009
  - NLP solvers sometimes have difficulty proving the optimality of a good point. The way they report that solution is with solver status "Terminated by Solver" and model status "Feasible Solution". SBB’s default behavior is to ignore such a solution (potentially go into failseq). With the new option acceptnonopt these solutions are accepted for further action.

- SBB offers node selections that switch between DFS and best bound/best estimate selection. In DFS mode SBB usually switches back to best bound/estimate if the DFS search resulted in a pruned or infeasible node or a new integer solution was found. In these cases it can be advantageous to search the close neighborhood of that node also in a DFS fashion. With the new option dfastay SBB is instructed to do some more DFS nodes even after a switch to best bound/estimate has been requested.

- January 11, 2002: Level 008
  - Maintenance release

- December 11, 2001: Level 007
  - NLP solvers sometimes have difficulty solving particular nodes and using up all the resources in this node. SBB provides options (see subres, subiter) to overcome these instances, but these options must be set in advance. SBB now keeps track of how much time is spent in the nodes, builds an average over time, and automatically controls the time spend in each node. The option avgresemult allows the user to customize this new feature.

- November 13, 2001: Level 006
  - Maintenance release

- May 22, 2001: Level 005
  - SBB derives an implicit, absolute termination tolerance if the model has a discrete objective row. This may speed up the overall time if the user has tight termination tolerances (optca, optcr).
SBB passes indices of rows with domain violations back to the LST file. All domain violation from the root node and from all sub nodes are reported, and the user can take advantage of this information to overcome these violations.

- March 21, 2001: Level 004
  - Pseudo Costs are available in SBB. Check section SBB with Pseudo Costs
  - A mix of DFS/Best Bound/Best Estimate node selection schemes are available through the \texttt{nodesel} option.
  - The \texttt{tryint} option now works for integer variables.
  - Additional information about node and variable selection can be printed to the Log file using the \texttt{printbbinfo} option.

- December 22, 2000: Level 003
  - MINOS and SNOPT are available as NLP subsolvers.

- November 19, 2000: Level 002
  - The model and solver status returned by SBB are synchronized with return codes reported by DICOPT.

- October 16, 2000: Level 001
  - SBB introduced to GAMS distribution 19.5.
  - CONOPT and CONOPT2 are the available NLP subsolvers.

2 Introduction

SBB is a GAMS solver for Mixed Integer Nonlinear Programming (MINLP) models. It is based on a combination of the standard Branch and Bound (B&B) method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. Currently, SBB can use

- CONOPT
- MINOS
- SNOPT

as solvers for submodels.

SBB supports all types of discrete variables supported by GAMS, including:

- Binary
- Integer
- Semicont
- Semiint
- Sos1
- Sos2
3 The Branch and Bound Algorithm

The Relaxed Mixed Integer Nonlinear Programming (RMINLP) model is initially solved using the starting point provided by the modeler. SBB will stop immediately if the RMINLP model is unbounded or infeasible, or if it fails (see option infeasseq and failseq below for an exception). If all discrete variables in the RMINLP model are integer, SBB will return this solution as the optimal integer solution. Otherwise, the current solution is stored and the Branch and Bound procedure will start.

During the Branch and Bound process, the feasible region for the discrete variables is subdivide, and bounds on discrete variables are tightened to new integer values to cut off the current non-integer solutions. Each time a bound is tightened, a new, tighter NLP submodel is solved starting from the optimal solution to the previous looser submodel. The objective function values from the NLP submodel is assumed to be lower bounds on the objective in the restricted feasible space (assuming minimization), even though the local optimum found by the NLP solver may not be a global optimum. If the NLP solver returns a Locally Infeasible status for a submodel, it is usually assumed that there is no feasible solution to the submodel, even though the infeasibility only has been determined locally (see option infeasseq below for an exception). If the model is convex, these assumptions will be satisfied and SBB will provide correct bounds. If the model is not convex, the objective bounds may not be correct and better solutions may exist in other, unexplored parts of the search space.

4 SBB with Pseudo Costs

Over the last decades quite a number of search strategies have been successfully introduced for mixed integer linear programming (for details see e.g. J.T. Linderoth and M.W.P. Savelsbergh, A Computational Study of Search Strategies for Mixed Integer Programming, INFORMS Journal on Computing, 11(2), 1999). Pseudo costs are key elements of sophisticated search strategies. Using pseudo costs, we can estimate the degradation of the objective function if we move a fractional variable to a close integer value. Naturally, the variable selection can be based on pseudo costs (see SBB option varsel). Node selection can also make use of pseudo cost: If we can estimate the change of the objective for moving one fractional variable to the closed integer value, we can then aggregate this change for all fractional variables, to estimate the objective of the best integer solution reachable from a particular node (see SBB option nodesel).

Unfortunately, the computation of pseudo cost can be a substantial part of the overall computation. Models with a large number of fractional variables in the root node are not good candidates for search strategies which require pseudo costs (varsel 3, nodesel 3,5,6). The impact (positive or negative) of using pseudo cost depends significantly on the particular model. At this stage, general statements are difficult to make.

Selecting pseudo cost related search strategies (varsel 3, nodesel 3,5,6) may use computation time which sometimes does not pay off. However, we encourage the user to try these options for difficult models which require a large number of branch-and-bound nodes to solve.

5 The SBB Options

SBB works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are iterlim, reallim, nodlim, optca, optcr, optfile, cheat, and cutoff. A description of all available GAMS options can be found in Chapter "Basic Solver Usage". GAMS options prioropt and tryint are also accepted by SBB.

SBB uses the var.prior information to select the fractional variable with the smallest priority during the variable selection process. SBB uses the tryint information to set the branching direction in the B&B algorithm. At the beginning, SBB looks at the levels of the discrete variables provided by the user and if abs(round(x.l)−x.l) < m.tryint, SBB will branch on that variable in the direction of round(x.l). For example, x.l = 0.9 and m.tryint = 0.2. We have abs(round(0.9) − 0.9) = 0.1 < 0.2, so when SBB decides to branch on this variable (because it is fractional, lets say with value 0.5), the node explored next will have the additional constraint x ≥ 1 (the node with x ≤ 0 will be explored later). If everything goes well (there is the chance that we end up in a different local optima in the subsolves for non-convex problems), SBB should reproduce a preset incumbent solution in a couple of nodes.

If you specify <modelname>.optfile = 1; before the SOLVE statement in your GAMS model, SBB will then look for and read an option file with the name sbb.opt (see "Using Solver Specific Options" for general use of solver option files).
Unless explicitly specified in the SBB option file, the NLP subsolvers will not read an option file. The syntax for the SBB option file is

```plaintext
optname value
```

with one option on each line.

For example,

```plaintext
rootsolver conopt.1
subsolver snopt
loginterval 10
```

The first two lines determine the NLP subsolvers for the Branch and Bound procedure. CONOPT with the option file conopt.opt will be used for solving the root node. SNOPT with no option file will be used for the remaining nodes. The last option determines the frequency for log line printing. Every 10th node, and each node with a new integer solution, causes a log line to be printed. The following options are implemented:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptnonopt</td>
<td>accepts feasible solution from subsolver</td>
<td>0</td>
</tr>
<tr>
<td>avgresmult</td>
<td>average resource multiplicator</td>
<td>5</td>
</tr>
<tr>
<td>dfsstay</td>
<td>keeps DFS node selection after solution has been found</td>
<td>0</td>
</tr>
<tr>
<td>epint</td>
<td>integer feasibility tolerance</td>
<td>1.0e-5</td>
</tr>
<tr>
<td>failseq</td>
<td>solver sequence for failed nodes</td>
<td></td>
</tr>
<tr>
<td>infeasseq</td>
<td>solver sequence for infeasible nodes</td>
<td></td>
</tr>
<tr>
<td>intsollim</td>
<td>maximum number of integer solutions</td>
<td>2100000000</td>
</tr>
<tr>
<td>loginterval</td>
<td>progress display interval</td>
<td>1</td>
</tr>
<tr>
<td>loglevel</td>
<td>level of solver display</td>
<td>1</td>
</tr>
<tr>
<td>memnodes</td>
<td>maximum number of nodes in memory</td>
<td>10000</td>
</tr>
<tr>
<td>nodepselector</td>
<td>node selection strategy</td>
<td>0</td>
</tr>
<tr>
<td>printbbinfo</td>
<td>prints additional node info</td>
<td>0</td>
</tr>
<tr>
<td>rootsolver</td>
<td>solver for the root node</td>
<td>GAMS NLP solver</td>
</tr>
<tr>
<td>solvelink</td>
<td>Solvelink for GAMS NLP solver</td>
<td>5</td>
</tr>
<tr>
<td>subiter</td>
<td>iteration limit for the subsolve</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>subres</td>
<td>resource limit for the subsolve</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>subsolver</td>
<td>solver for the subproblems</td>
<td>GAMS NLP solver</td>
</tr>
<tr>
<td>varsel</td>
<td>variable selection strategy at each node</td>
<td>0</td>
</tr>
</tbody>
</table>

**acceptnonopt (integer):** accepts feasible solution from subsolver

If this option is set to 1 and the subsolver terminates with solver status *Terminated by Solver* and model status *Intermediate Nonoptimal* SBB takes this as a good solution and keeps on going. In default mode such a return is treated as a subsolver failure and the **failseq** is consulted.

(default = 0)

**avgresmult (integer):** average resource multiplicator

Similar to **subres**, this option allows the user to control the time limit spend in a node. SBB keeps track of how much time is spent in the nodes, and builds an average over time. This average multiplied by the factor **avgresmult** is set as a time limit for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure: the node is ignored or the solvers in the **failseq** are called. The default multiplier **avgresmult** is 5. Setting **avgresmult** to 0 will disable the automatic time limit feature. A multiplier is not very useful for very small node solution times; therefore, independent of each node, SBB grants the solver at
least 5 seconds to solve the node. The competing option subres overwrites the automatically generated resource limit.
(default = 5)

**dfsstay (integer):** keeps DFS node selection after solution has been found

If the node selection is a B*/DFS mix, SBB switches frequently to DFS node selection mode. It switches back into B* node selection mode, if no subnodes were created (new int, pruned, infeasible, fail). It can be advantageous to search the neighborhood of the last node also in a DFS manner. Setting dfsstay to \( n \) instructs SBB to stay in DFS mode for another \( n \) nodes.

(default = 0)

**epint (real):** integer feasibility tolerance

The integer infeasibility tolerance.

Range: \([1e-9, 1]\)

(default = 1.0e-5)

**failseq (string):** solver sequence for failed nodes

\[\text{solver1}[^{.n1}] \text{ solver2}[^{.n2}] \ldots\] where \( \text{solver1} \) is the name of a GAMS NLP solver to be used if the default solver fails, i.e., if it was not stopped by an iteration, resource, or domain limit and does not return a locally optimal or locally infeasible solution. \( .n1 \) is the value of optfile passed to the alternative NLP solver. If \( .n1 \) is left blank it is interpreted as zero. Similarly, \( \text{solver2} \) is the name of a GAMS NLP solver that is used if \( \text{solver1} \) fails, and \( .n2 \) is the value of optfile passed to the second NLP solver. If you have a difficult model where solver failures are not unlikely, you may add more \( \text{solver}.n \) pairs. You can use the same solver several times with different options files. \( \text{failseq conopt conopt.2 conopt.3} \) means to try CONOPT with no options file. If this approach also fails, try CONOPT with options file \( \text{conopt.op2} \), and if it again fails, try CONOPT with options file \( \text{conopt.op3} \). If all solver and options file combinations fail the node will be labeled ignored and the node will not be explored further. The default is to try only one solver (the rootsolver or subsolver) and to ignore nodes with a solver failure.

**infeasseq (string):** solver sequence for infeasible nodes

\[\text{level} \text{ solver1}[^{.n1}] \text{ solver2}[^{.n2}] \ldots\] The purpose of infeasseq is to avoid cutting parts of the search tree that appear to be infeasible but really are feasible. If the NLP solver labels a node Locally Infeasible and the model is not convex a feasible solution may actually exist. If SBB is high in the search tree it can be very drastic to prune the node immediately. SBB is therefore directed to try the solver/option combinations in the list as long as the depth in the search tree is less than the integer value \( \text{level} \). If the list is exhausted without finding a feasible solution, the node is assumed to be infeasible. The default is to trust that Locally Infeasible nodes are indeed infeasible and to remove them from further consideration.

**intsollim (integer):** maximum number of integer solutions

Maximum number of integer solutions. If this number is exceeded, SBB will terminate and return the best solution found so far.

(default = 2100000000)

**loginterval (integer):** progress display interval

The interval (number of nodes) for which log lines are written.

(default = 1)

**loglevel (integer):** level of solver display

The level of log output.

(default = 1)

0 only SBB log lines with one line every loginterval nodes
1 NLP solver log for the root node plus SBB loglines as 0
2 NLP solver log for all nodes plus SBB log lines as 0
memnodes (integer): maximum number of nodes in memory

The maximum number of nodes SBB can have in memory. If this number is exceeded, SBB will terminate and return the best solution found so far.
(default = 10000)

nodesel (integer): node selection strategy

Node selection scheme.
(default = 0)

0 automatic
1 Depth First Search (DFS)
2 Best Bound (BB)
3 Best Estimate (BE)
4 DFS/BB mix
5 DFS/BE mix
6 DFS/BB/BE mix

printbbinfo (integer): prints additional node info

Additional info of log output.
(default = 0)

0 print no additional info
1 print variable selection letter The node and variable selection for the current node are indicated by a two letter code at the end of the log line. The first letter represents the node selection: D for DFS, B for Best Bound, and E for Best Estimate. The second letter represents the variable selection: X for maximum infeasibility, N for minimum infeasibility, and P for pseudo cost.
2 print best estimate

rootsolver (string): solver for the root node

solver[n] Solver is the name of the GAMS NLP solver that should be used in the root node, and n is the integer corresponding to optfile for the root node. If .n is missing, the optfile treated as zero i.e. the NLP solver will not look for an options file. This SBB option can be used to overwrite the default that uses the NLP solver specified with an Option NLP = solver; statement or the default GAMS solver for NLP.
(default = GAMS NLP solver)

solvelink (integer): Solvelink for GAMS NLP solver

(default = 5)

1 Call GAMS NLP solver via script
2 Call GAMS NLP solver via module
5 Call GAMS NLP solver in memory

subiter (integer): iteration limit for the subsolve

The default for subiter passed on through iterlim. Similar to subres but sets the iteration limit for solving a node in the B&B tree.
(default = GAMS iterlim)

subres (real): resource limit for the subsolve

The default for subres passed on through reslim. Sets the time limit in seconds for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure and the node is ignored, or the solvers in the failseq are called.
(default = GAMS reslim)
subsolver (string): solver for the subproblems

solver[n] Similar to root solver but applied to the subnodes.
(default = GAMS NLP solver)

varsel (integer): variable selection strategy at each node

Variable selection scheme.
(default = 0)

0 automatic
1 maximum integer infeasibility
2 minimum integer infeasibility
3 pseudo costs

6 The SBB Log File

The SBB Log file (usually directed to the screen) can be controlled with the loginterval and loglevel options in SBB. It will by default first show the iteration output from the NLP solver that solves the root node. This is followed by output from SBB describing the search tree. An example of this search tree output follows:

Root node solved locally optimal.

<table>
<thead>
<tr>
<th>Node</th>
<th>Act</th>
<th>Lev</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Int.</th>
<th>Best Bound</th>
<th>Gap</th>
<th>(2 secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8457.6878</td>
<td>3</td>
<td>-</td>
<td>8457.6878</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8491.2869</td>
<td>2</td>
<td>-</td>
<td>8457.6878</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>8518.1779</td>
<td>1</td>
<td>-</td>
<td>8457.6878</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
* 3    | 3   | 3   | 9338.1020 | 0    | 9338.1020 | 8457.6878  | 0.1041 |
| 4    | 2   | 1   | pruned    | -    | 9338.1020 | 8491.2869  | 0.0997 |

Solution satisfies optcr

Statistics:
Iterations : 90
NLP Seconds : 0.110000
B&B nodes : 3
MIP solution : 9338.101979 found in node 3
Best possible : 8491.286941
Absolute gap : 846.815039 optca : 0.000000
Relative gap : 0.099728 optcr : 0.100000
Model Status : 8
Solver Status : 1

NLP Solver Statistics
Total Number of NLP solves : 7
Total Number of NLP failures: 0
Details: conopt
# execs 7
# failures 0
Terminating.

The fields in the log are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>The number of the current node. The root node is node 0.</td>
</tr>
<tr>
<td>Act</td>
<td>The number of active nodes defined as the number of subnodes that have not yet been solved.</td>
</tr>
<tr>
<td>Lev</td>
<td>The level in the search tree, i.e., the number of branches needed to reach this node.</td>
</tr>
</tbody>
</table>
The objective function value for the node. A numerical value indicates that the node was solved and the objective was good enough for the node to not be ignored. "pruned" indicates that the objective value was worse than the Best Integer value, "infeasible" indicates that the node was Infeasible or Locally Infeasible, and "ignored" indicates that the node could not be solved (see under failseq above).

The number of integer infeasibilities, i.e. the number of variables that are supposed to be binary or integer that do not satisfy the integrality requirement. Semi continuous variables and SOS variables may also contribute to IInf.

The value of the best integer solution found so far. A dash (-) indicates that an integer solution has not yet been found. A star (*) in column one indicates that the node is integer and that the solution is better than the best yet found.

The minimum value of "Objective" for the subnodes that have not been solved yet (maximum for maximization models). For convex models, Best Bound will increase monotonically. For nonconvex models, Best Bound may decrease, indicating that the Objective value for a node was not a valid lower bound for that node.

The relative gap between the Best Integer solution and the Best Bound.

The remaining part of the Log file displays various solution statistics similar to those provided by the MIP solvers. This information can also be found in the Solver Status area of the GAMS listing file.

The following Log file shows cases where the NLP solver fails to solve a subnode. The text "ignored" in the Objective field shows the failure, and the values in parenthesis following the Gap field are the Solve and Model status returned by the NLP solver:

The next Log file shows the effect of the infeasseq and failseq options on the model above. CONOPT with options file conopt.opt (the default solver and options file pair for this model) considers the first subnode to be locally infeasible. CONOPT1, MINOS, and SNOPT, all with no options file, are therefore tried in sequence. In this case, they all declare the node infeasible and it is considered to be infeasible.

In node 3, CONOPT fails but CONOPT1 finds a Locally Optimal solution, and this solution is then used for further search. The option file for the following run would be:

```
rootsolver conopt.1
subsolver conopt.1
infeasseq conopt1 minos snopt
```

The log looks as follows:
Executing snopt

1 1 1 infeasible - - 6046.0186 -
2 0 1 6042.0995 10 - 6042.0995 -

conopt.1 failed. 4 TERMINATED BY SOLVER, 7 FEASIBLE SOLUTION

Executing conopt1

3 1 2 4790.2373 8 - 6042.0995 -
4 2 3 4481.4156 6 - 6042.0995 -

conopt.1 reports locally infeasible

Executing conopt1

conopt1 reports locally infeasible

Executing minos

minos failed. 4 TERMINATED BY SOLVER, 6 INTERMEDIATE INFEASIBLE

Executing snopt

5 3 4 infeasible - - 6042.0995 -
6 2 4 4480.3778 4 - 6042.0995 -

The Log file shows a solver statistic at the end, summarizing how many times an NLP was executed and how often it failed:

NLP Solver Statistics
Total Number of NLP solves : 45
Total Number of NLP failures: 13
Details: conopt minos snopt
# execs 34 3 8
# failures 4 3 6

The solutions found by the NLP solver to the subproblems in the Branch and Bound may not be the global optima. Therefore, the objective can improve even though we restrict the problem by tightening some bounds. These jumps of the objective in the wrong direction which might also have an impact on the best bound/possible are reported in a separate statistic:

Non convex model!

# jumps in best bound : 2
Maximum jump in best bound : 20.626587 in node 13
# jumps to better objective : 2
Maximum jump in objective : 20.626587 in node 13

7 Comparison of DICOPT and SBB

Until recently, MINLP models could only be solved with the DICOPT solver. DICOPT is based on the outer approximation method. Initially, the RMINLP model is solved just as in SBB. The model is then linearized around this point and a linear MIP model is solved. The discrete variables are then fixed at the optimal values from the MIP model, and the resulting NLP model is solved. If the NLP model is feasible, we have an integer feasible solution.

The model is linearized again and a new MIP model with both the old and new linearized constraints is solved. The discrete variables are again fixed at the optimal values, and a new NLP model is solved.

The process stops when the MIP model becomes infeasible, when the NLP solution becomes worse, or, in some cases, when bounds derived from the MIP model indicate that it is safe to stop.

DICOPT is based on the assumption that MIP models can be solved efficiently while NLP models can be expensive and difficult to solve. The MIP models try to approximate the NLP model over a large area and solve it using cheap linear technology. Ideally, only a few NLPs must be solved.

DICOPT can experience difficulties solving models, if many or all the NLP submodels are infeasible. DICOPT can also have problems if the linearizations used for the MIP model create ill-conditioned models. The MIP models may become very difficult to solve, and the results from the MIP models may be poor as initial values for the NLP models. The linearized constraint used by DICOPT may also exclude certain areas of the feasible space from consideration.
SBB uses different assumptions and works very differently. Most of the work in SBB involves solving NLP models. Since the NLP submodels differ only in one or a few bounds, the assumption is that the NLP models can be solved quickly using a good restart procedure. Since the NLP models differ very little and good initial values are available, the solution process will be fairly reliable compared to the solution process in DICOPT, where initial values of good quality seldom are available. Because search space is reduced based on very different grounds than in DICOPT, other solutions may therefore be explored. Overall, DICOPT should perform better on models that have a significant and difficult combinatorial part, while SBB may perform better on models that have fewer discrete variables but more difficult nonlinearities (and possibly also on models that are fairly non convex).
1 Introduction

This section describes the GAMS interface to the general-purpose NLP solver SNOPT, (Sparse Nonlinear Optimizer) which implements a sequential quadratic programming (SQP) method for solving constrained optimization problems with smooth nonlinear functions in the objective and constraints. The optimization problem is assumed to be stated in the form

\[
\begin{align*}
\text{minimize or maximize } & f_0(x) \\
f(x) & \sim b_1 \\
\text{subject to} & \quad A_L x \sim b_2 \\
& \quad l \leq x \leq u,
\end{align*}
\]

where \(x \in \mathbb{R}^n\), \(f_0(x)\) is a linear or nonlinear smooth objective function, \(l\) and \(u\) are constant lower and upper bounds, \(f(x)\) is a set of nonlinear constraint functions, \(A_L\) is a sparse matrix, \(~\) is a vector of relational operators (\(\leq, \geq\) or \(=\)), and \(b_1\) and \(b_2\) are right-hand side constants. \(f(x) \sim b_1\) are the nonlinear constraints of the model and \(A_L x \sim b_2\) form the linear constraints.

The gradients of \(f_0\) and \(f_i\) are automatically provided by GAMS, using its automatic differentiation engine.

The bounds may include special values \(-\text{INF}\) or \(+\text{INF}\) to indicate \(l_j = -\infty\) or \(u_j = +\infty\) for appropriate \(j\). Free variables have both bounds infinite and fixed variables have \(l_j = u_j\).

SNOPT uses sequential quadratic programming (SQP) algorithms that obtain search directions from a sequence of quadratic programming subproblems. Each QP subproblem minimizes a quadratic model of a certain Lagrangian function subject to a linearization of the constraints. An augmented Lagrangian merit function is reduced along each search direction to ensure convergence from any starting point.

SNOPT is most efficient if only some of the variables enter nonlinearly, or if the number of active constraints (including simple bounds) is nearly as large as the number of variables. SNOPT requires relatively few evaluations of the problem functions.

1.1 Problem Types

If the nonlinear functions are absent, the problem is a linear program (LP) and SNOPT applies the primal simplex method [2]. Sparse basis factors are maintained by LUSOL [7] as in MINOS [13].

If only the objective is nonlinear, the problem is linearly constrained (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). Note that GAMS models have an objective variable instead of an objective function. The GAMS/SNOPT link will try to substitute out the objective variable and reformulate the model such that SNOPT will see a true objective function.

For both linearly and nonlinearly constrained problems SNOPT applies a sparse sequential quadratic programming (SQP) method [10] using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver NPSOL [6, 9].

In general, SNOPT requires less matrix computation than NPSOL and fewer evaluations of the functions than the nonlinear algorithms in MINOS [14, 15]. It is suitable for nonlinear problems with thousands of constraints and variables, and is most efficient if only some of the variables enter nonlinearly, or there are relatively few degrees of freedom at a solution (i.e., many constraints are active). However, unlike previous versions of SNOPT, there is no limit on the number of degrees of freedom.

1.2 Selecting the SNOPT Solver

The GAMS system can be instructed to use the SNOPT solver by incorporating the following option in the GAMS model:

```plaintext
option NLP=SNOPT;
```

If the model contains non-smooth functions like \(\text{abs}(x)\), or \(\text{max}(x,y)\) you can try to get it solved by SNOPT using

```plaintext
option DNLP=SNOPT;
```
These models have discontinuous derivatives however, and SNOPT was not designed for solving such models. Discontinuities in the gradients can sometimes be tolerated if they are not too close to an optimum.

It is also possible to specify NLP=SNOPT or DNLP=SNOPT on the command line, as in:

```plaintext
> gamslib chem
> gams chem nlp=snopt
```

In the Windows IDE command line parameters can be specified in the parameter combo box above the edit window.

## 2 Description of the method

Here we briefly describe the main features of the SQP algorithm used in SNOPT and introduce some terminology. The SQP algorithm is fully described by Gill, Murray and Saunders [11].

### 2.1 Objective function reconstruction

The first step GAMS/SNOPT performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. SNOPT however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```plaintext
obj.. z =e= sum(i, sqr(r(i)));
model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form (1) by saying minimize \( z \) subject to \( z = \sum_i r_i^2 \) and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize the nonlinear expression \( \sum_i r_i^2 \) directly. This can be achieved by a simple reformulation: \( z \) can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable \( z \) is a free continuous variable (no bounds are defined on \( z \)),
- \( z \) appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- \( z \) is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by SNOPT. For instance the model chem.gms from the model library solves in 16 iterations. When we add the bound

```plaintext
energy.lo = 0;
```

on the objective variable energy and thus preventing it from being substituted out, SNOPT will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```plaintext
variables x,y,z;
equations e1,e2;
e1..z =e= y;
e2..y =e= sqr(1+x);
model m /all/;
option nlp=snopt;
solve m using nlp minimizing z;
```
will be reformulated as an *unconstrained* optimization problem
\[
\min f(x) = (1+x)^2.
\]
These additional reformulations can be turned off by using the statement option `reform = 0`; (See Options).

### 2.2 Constraints and slack variables

Problem (1) contains \( n \) variables in \( x \). Let \( m \) be the number of components of \( f(x) \) and \( A_Lx \) combined. The upper and lower bounds on those terms define the general constraints of the problem. SNOPT converts the general constraints to equalities by introducing a set of slack variables \( s = (s_1, s_2, \ldots, s_m)^T \). For example, the linear constraint \( 5 \leq 2x_1 + 3x_2 \leq +\infty \) is replaced by \( 2x_1 + 3x_2 - s_1 = 0 \) together with the bounded slack \( 5 \leq s_1 \leq +\infty \). Problem (1) can be written in the equivalent form
\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f(x) - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u.
\end{align*}
\]

where a maximization problem is cast into a minimization by multiplying the objective function by \(-1\).

The linear and nonlinear general constraints become equalities of the form \( f(x) - s_N = 0 \) and \( A_Lx - s_L = 0 \), where \( s_L \) and \( s_N \) are known as the *linear* and *nonlinear* slacks.

### 2.3 Major iterations

The basic structure of SNOPT's solution algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates \( \{x_k\} \) that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate \( \{x_k\} \) a QP subproblem is used to generate a search direction towards the next iterate \( \{x_{k+1}\} \). The constraints of the subproblem are formed from the linear constraints \( A_Lx - s_L = 0 \) and the nonlinear constraint linearization
\[
f(x_k) + f'(x_k)(x - x_k) - s_N = 0,
\]
where \( f'(x_k) \) denotes the *Jacobian*: a matrix whose rows are the first derivatives of \( f(x) \) evaluated at \( x_k \). The QP constraints therefore comprise the \( m \) linear constraints
\[
\begin{align*}
f'(x_k)x & \quad -s_N = -f(x_k) + f'(x_k)x_k, \\
A_Lx & \quad -s_L = 0,
\end{align*}
\]
where \( x \) and \( s \) are bounded by \( l \) and \( u \) as before. If the \( m \times n \) matrix \( A \) and \( m \)-vector \( b \) are defined as
\[
A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \text{and } b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix},
\]
then the QP subproblem can be written as
\[
\begin{align*}
\text{QP}_k & \quad \min_{x,s} q(x,x_k) = g_k^T (x - x_k) + \frac{1}{2} (x - x_k)^T H_k (x - x_k) \\
\end{align*}
\]
subject to \( Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u,
\]
where \( q(x,x_k) \) is a quadratic approximation to a modified Lagrangian function [10]. The matrix \( H_k \) is a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration. If some of the variables enter the Lagrangian linearly the Hessian will have some zero rows and columns. If the nonlinear variables appear first, then only the leading \( n_1 \) rows and columns of the Hessian need be approximated, where \( n_1 \) is the number of nonlinear variables.
2.4 Minor iterations

Solving the QP subproblem is itself an iterative procedure. Here, the iterations of the QP solver SQOPT [12] form the minor iterations of the SQP method.

SQOPT uses a reduced-Hessian active-set method implemented as a reduced-gradient method similar to that in MINOS [14]. At each minor iteration, the constraints $Ax - s = b$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

where the basis matrix $B$ is square and nonsingular and the matrices $S, N$ are the remaining columns of $(A-I)$. The vectors $x_B, x_S, x_N$ are the associated basic, superbasic, and nonbasic variables components of $(x,s)$.

The term active-set method arises because the nonbasic variables $x_N$ are temporarily frozen at their upper or lower bounds, and their bounds are considered to be active. Since the general constraints are satisfied also, the set of active constraints takes the form

$$
\begin{pmatrix}
B & S & N \\
& & 1
\end{pmatrix}
\begin{pmatrix}
x_B \\
x_S \\
x_N
\end{pmatrix} =
\begin{pmatrix}
b \\
0
\end{pmatrix},
$$

where $x_N$ represents the current values of the nonbasic variables. (In practice, nonbasic variables are sometimes frozen at values strictly between their bounds.) The reduced-gradient method chooses to move the superbasic variables in a direction that will improve the objective function. The basic variables "tag along" to keep $Ax - s = b$ satisfied, and the nonbasic variables remain unaltered until one of them is chosen to become superbasic.

At a nonoptimal feasible point $(x,s)$ we seek a search direction $p$ such that $(x,s) + p$ remains on the set of active constraints yet improves the QP objective. If the new point is to be feasible, we must have $Bp_B + Sp_S + Np_N = 0$ and $p_N = 0$. Once $p_S$ is specified, $p_B$ is uniquely determined from the system $Bp_B = -Sp_S$. It follows that the superbasic variables may be regarded as independent variables that are free to move in any desired direction. The number of superbasic variables ($n_S$ say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, $n_S$ is a measure of how nonlinear the problem is. In particular, $n_S$ need not be more than one for linear problems.

2.5 The reduced Hessian and reduced gradient

The dependence of $p$ on $p_S$ may be expressed compactly as $p = Zp_S$, where $Z$ is a matrix that spans the null space of the active constraints:

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}$$

where $P$ permutes the columns of $(A-I)$ into the order $(BSN)$. Minimizing $q(x,x_k)$ with respect to $p_S$ now involves a quadratic function of $p_S$:

$$g^T Z p_S + \frac{1}{2} p_S^T Z^T H Z p_S,$$

where $g$ and $H$ are expanded forms of $g_k$ and $H_k$ defined for all variables $(x,s)$. This is a quadratic with Hessian $Z^T H Z$ (the reduced Hessian) and constant vector $Z^T g$ (the reduced gradient). If the reduced Hessian is nonsingular, $p_S$ is computed from the system

$$Z^T H Z p_S = -Z^T g.$$  \hspace{1cm} (5)

The matrix $Z$ is used only as an operator, i.e., it is not stored explicitly. Products of the form $Zv$ or $Z^T g$ are obtained by solving with $B$ or $B^T$. The package LUSOL [7] is used to maintain sparse $LU$ factors of $B$ as the $BSN$ partition changes. From the definition of $Z$, we see that the reduced gradient can be computed from

$$B^T \pi = g_B, \quad Z^T g = g_S - S^T \pi,$$

where $\pi$ is an estimate of the dual variables associated with the $m$ equality constraints $Ax - s = b$, and $g_B$ is the basic part of $g$. 

SNOPT 493
By analogy with the elements of $Z^T g$, we define a vector of reduced gradients (or reduced costs) for all variables in $(x, s)$:

$$
d = g - \begin{pmatrix} A^T \\ I \end{pmatrix} \pi,
$$

so that $d_S = Z^T g$.

At a feasible point, the reduced gradients for the slacks $s$ are the dual variables $\pi$.

The optimality conditions for subproblem QP \( k \) (2) may be written in terms of $d$. The current point is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for all superbasic variables ($d_S = 0$). In practice, SNOPT requests an approximate QP solution $(\hat{x}_k, \hat{s}_k, \pi_k)$ with slightly relaxed conditions on $d_j$.

If $d_S = 0$, no improvement can be made with the current BSN partition, and a nonbasic variable with non-optimal reduced gradient is selected to be added to $S$. The iteration is then repeated with $n_S$ increased by one. At all stages, if the step $(x, s) + a p$ would cause a basic or superbasic variable to violate one of its bounds, a shorter step $(x, s) + \alpha p$ is taken, one of the variables is made nonbasic, and $n_S$ is decreased by one.

The process of computing and tested reducing gradients $d_N$ is known as pricing (a term introduced in the context of the simplex method for linear programming). Pricing the $j$th variable means computing $d_j = g_j - a_j^T \pi$, where $a_j$ is the $j$th column of $(A - I)$. If there are significantly more variables than general constraints (i.e., $n \gg m$), pricing can be computationally expensive. In this case, a strategy known as partial pricing can be used to compute and test only a subset of $d_N$.

Solving the reduced Hessian system (5) is sometimes expensive. With the option QPSolver Cholesky, an upper-triangular matrix $R$ is maintained satisfying $R^T R = Z^T H Z$. Normally, $R$ is computed from $Z^T H Z$ at the start of phase 2 and is then updated as the BSN sets change. For efficiency the dimension of $R$ should not be excessive (say, $n_S \leq 1000$). This is guaranteed if the number of nonlinear variables is "moderate". Other QPSolver options are available for problems with many degrees of freedom.

### 2.6 The merit function

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f(x) - \pi^T (F(x) - s_N) + \frac{1}{2} (F(x) - s_N)^T D (F(x) - s_N),$$

where $D$ is a diagonal matrix of penalty parameters. If $(x_k, s_k, \pi_k)$ denotes the current solution estimate and $(\hat{x}_k, \hat{s}_k, \pi_k)$ denotes the optimal QP solution, the linesearch determines a step $\alpha_k$ ($0 < \alpha_k \leq 1$) such that the new point

$$
\begin{pmatrix}
  x_{k+1} \\
  s_{k+1} \\
  \pi_{k+1}
\end{pmatrix} =
\begin{pmatrix}
  x_k \\
  s_k \\
  \pi_k
\end{pmatrix} + \alpha_k
\begin{pmatrix}
  \hat{x}_k - x_k \\
  \hat{s}_k - s_k \\
  \pi_k - \pi_k
\end{pmatrix}
$$

(7)

gives a sufficient decrease in the merit function. When necessary, the penalties in $D$ are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ [9]. As in NPSOL, $s_N$ is adjusted to minimize the merit function as a function of $s$ prior to the solution of the QP subproblem. For more details, see [3, 6].

### 2.7 Treatment of constraint infeasibilities

SNOPT makes explicit allowance for infeasible constraints. Infeasible linear constraints are detected first by solving a problem of the form

<table>
<thead>
<tr>
<th>FLP</th>
<th>minimize $e^T (v + w)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>subject to $\ell \leq (A_L x - v + w) \leq u$, $v \geq 0$, $w \geq 0$,</td>
</tr>
</tbody>
</table>
where $e$ is a vector of ones. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called elastic programming. We also describe it as minimizing the $\ell_1$ norm of the infeasibilities.)

If the linear constraints are infeasible ($v \neq 0$ or $w \neq 0$), SNOPT terminates without computing the nonlinear functions.

If the linear constraints are feasible, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve NP (1) as given, using search directions obtained from a sequence of quadratic programming subproblems (2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables $\pi$ for the nonlinear constraints become large), SNOPT enters “elastic” mode and solves the problem

$$
\text{NP}(\gamma) \quad \text{minimize} \quad f_0(x) + \gamma e^T (v + w) \\
\text{subject to} \quad \ell \leq \begin{pmatrix} x \\ f(x) - v + w \\ A_L x \end{pmatrix} \leq u, \, v \geq 0, \, w \geq 0,
$$

where $\gamma$ is a nonnegative parameter (the elastic weight), and $f(x) + \gamma e^T (v + w)$ is called a composite objective. If $\gamma$ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar $\ell_1$ formulation of NP is fundamental to the $S\ell_1$QP algorithm of Fletcher [4]. See also Conn [1].

The initial value of $\gamma$ is controlled by the optional parameter Elastic weight.

### 3 Starting points and advanced bases

A good starting point may be essential for solving nonlinear models. We show how such a starting point can be specified in a GAMS environment, and how SNOPT will use this information.

A related issue is the use of "restart" information in case a number of related models is solved in a row. Starting from an optimal point from a previous solve statement is in such situations often beneficial. In a GAMS environment this means reusing primal and dual information, which is stored in the .L and .M fields of variables and equations.

#### 3.1 Starting points

To specify a starting point for SNOPT use the .L level values in GAMS. For example, to set all variables $x_{i,j} := 1$ use $x.l(i,j)=1;$. The default values for level values are zero.

Setting a good starting point can be crucial for getting good results. As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points $(x_i, y_i)$:

**Example**

$$
\text{minimize} \quad r \\
\text{subject to} \quad (x_i - a)^2 + (y_i - b)^2 \leq r^2, \quad r \geq 0.
$$

This problem can be modeled in GAMS as follows.

```gams
set i points /p1*p10/;

parameters
    x(i)    x coordinates,
    y(i)    y coordinates;

* fill with random data
x(i) = uniform(1,10);
```
y(i) = uniform(1,10);

variables
    a  x coordinate of center of circle
    b  y coordinate of center of circle
    r  radius;

equations
    e(i)  points must be inside circle;

    e(i)..  sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);

r.lo = 0;

model m /all/;
option nlp=snopt;
solve m using nlp minimizing r;

Without help, SNOPT will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

\[
\begin{align*}
    x_{\text{min}} &= \min_i x_i, \\
    y_{\text{min}} &= \min_i y_i, \\
    x_{\text{max}} &= \max_i x_i, \\
    y_{\text{max}} &= \max_i y_i,
\end{align*}
\]

then good estimates are

\[
\begin{align*}
    a &= \frac{(x_{\text{min}} + x_{\text{max}})}{2}, \\
    b &= \frac{(y_{\text{min}} + y_{\text{max}})}{2}, \\
    r &= \sqrt{(a-x_{\text{min}})^2 + (b-y_{\text{min}})^2}.
\end{align*}
\]

Thus we include in our model:

parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
 ymin = smin(i, x(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );

and now the model solves very easily.

Level values can also be set implicitly as a result of assigning bounds. When a variable is bounded away from zero, for instance by the statement Y.LO = 1;, the SOLVE statement will override the default level of zero of such a variable in order to make it feasible.

Note: another way to formulate the model would be to minimize \( r^2 \) instead of \( r \). This allows SNOPT to solve the problem even with the default starting point.
3.2 Advanced basis

GAMS automatically passes on level values and basis information from one solve to the next. Thus, when we have two solve statements in a row, with just a few changes in between SNOPT will typically need very few iterations to find an optimal solution in the second solve. For instance, when we add a second solve to the fawley.gms model from the model library:

```gams
Model exxon /all/;
...
Solve exxon maximizing profit using lp;
Solve exxon maximizing profit using lp;
```

we observe the following iteration counts:

```
SOLVE SUMMARY
MODEL  exxon     OBJECTIVE profit
TYPE   LP        DIRECTION MAXIMIZE
SOLVER SNOPT     FROM LINE 278

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 1 OPTIMAL
**** OBJECTIVE VALUE  2899.2528

RESOURCE USAGE, LIMIT  0.016  1000.000
ITERATION COUNT, LIMIT 24       10000

....
```

```
SOLVE SUMMARY
MODEL  exxon     OBJECTIVE profit
TYPE   LP        DIRECTION MAXIMIZE
SOLVER SNOPT     FROM LINE 279

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 1 OPTIMAL
**** OBJECTIVE VALUE  2899.2528

RESOURCE USAGE, LIMIT  0.000  1000.000
ITERATION COUNT, LIMIT 0       10000
```

The first solve takes 24 iterations, while the second solve needs exactly zero iterations.

Basis information is passed on using the marginals of the variables and equations. In general the rule is:

- \( X.M = 0 \) basic
- \( X.M \neq 0 \) nonbasic if level value is at bound, superbasic otherwise

A marginal value of \( \text{EPS} \) means that the numerical value of the marginal is zero, but that the status is nonbasic or superbasic. The user can specify a basis by assigning zero or nonzero values to the \( .M \) values. It is further noted that if too many \( .M \) values are zero, the basis is rejected. This happens for instance when two subsequent models are too different. This decision is made based on the value of the \texttt{bratio} option (see Section Options).
4 Options

In many cases NLP models can be solved with GAMS/SNOPT without using solver options. For special situations it is possible to specify non-standard values for some or all of the options.

4.1 Printing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>major print level</td>
<td>Amount of information printed during optimization (listing file)</td>
<td>1</td>
</tr>
<tr>
<td>minor print level</td>
<td>Amount of information printed during optimization (listing file)</td>
<td>1</td>
</tr>
<tr>
<td>print frequency</td>
<td>Number of iterations between each log line (listing file)</td>
<td>100</td>
</tr>
<tr>
<td>solution</td>
<td>Prints SNOPT solution (listing file)</td>
<td>NO</td>
</tr>
<tr>
<td>summary frequency</td>
<td>Number of iterations between each log line (log file)</td>
<td>100</td>
</tr>
<tr>
<td>suppress parameters</td>
<td>Suppress printing of parameters (listing file)</td>
<td></td>
</tr>
<tr>
<td>system information</td>
<td>Provides additional information on the progress of the iterations (listing file)</td>
<td>NO</td>
</tr>
<tr>
<td>timing level</td>
<td>Amount of timing information (listing file)</td>
<td>3</td>
</tr>
</tbody>
</table>

4.2 Problem specification

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>feasible point</td>
<td>Ignore objective function and find a feasible point</td>
<td></td>
</tr>
<tr>
<td>infinite bound</td>
<td>Bounds larger than this number are considered Infinity</td>
<td>1.0e20</td>
</tr>
</tbody>
</table>

4.3 Convergence tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>major feasibility tolerance</td>
<td>Specifies how accurately the nonlinear constraints should be satisfied</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>major optimality tolerance</td>
<td>Specifies the final accuracy of the dual variables</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>minor feasibility tolerance</td>
<td>Feasibility tolerance applied to all variables and linear constraints</td>
<td>1.0e-6</td>
</tr>
</tbody>
</table>

4.4 Derivative checking

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>start constraint check</td>
<td>Can be used to reduce the range of finite-difference checks</td>
<td>1</td>
</tr>
<tr>
<td>start objective check</td>
<td>Can be used to reduce the range of finite-difference checks</td>
<td>1</td>
</tr>
<tr>
<td>stop constraint check</td>
<td>Can be used to reduce the range of finite-difference checks</td>
<td>MAXINT</td>
</tr>
<tr>
<td>stop objective check</td>
<td>Can be used to reduce the range of finite-difference checks</td>
<td>MAXINT</td>
</tr>
<tr>
<td>verify level</td>
<td>Finite-difference checks on the derivatives</td>
<td>0</td>
</tr>
</tbody>
</table>

4.5 Scaling

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale option</td>
<td>Scaling of linear/nonlinear variables</td>
<td>1</td>
</tr>
</tbody>
</table>
### 4.6 Other tolerances

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash tolerance</td>
<td>Allow crash procedure to ignore small elements in eligible columns</td>
<td>0.1</td>
</tr>
<tr>
<td>linesearch tolerance</td>
<td>Accuracy required for steplength</td>
<td>0.1</td>
</tr>
<tr>
<td>pivot tolerance</td>
<td>Used to prevent columns entering the basis making it almost singular</td>
<td>3.7e-11</td>
</tr>
</tbody>
</table>

### 4.7 QP subproblems

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash option</td>
<td>Controls the basis crash algorithm</td>
<td>0 or 3</td>
</tr>
<tr>
<td>elastic weight</td>
<td>Used only during elastic mode</td>
<td>1.0e4</td>
</tr>
<tr>
<td>iterations limit</td>
<td>Minor iteration limit (ITERLIM)</td>
<td>1000</td>
</tr>
<tr>
<td>partial price</td>
<td>Number of segments in partial pricing strategy</td>
<td>10</td>
</tr>
<tr>
<td>qpsolver</td>
<td>QP Solver</td>
<td>Cholesky</td>
</tr>
</tbody>
</table>

### 4.8 SQP method

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>central difference interval</td>
<td>Not applicable: GAMS provides analytic derivatives</td>
<td>6.0e-6</td>
</tr>
<tr>
<td>cold start</td>
<td>Ignore advanced basis and use CRASH procedure</td>
<td></td>
</tr>
<tr>
<td>derivative level</td>
<td>Specifies which derivatives are provided</td>
<td>3</td>
</tr>
<tr>
<td>derivative linesearch</td>
<td>Linesearch method (safeguarded cubic interpolation) with use of derivatives</td>
<td></td>
</tr>
<tr>
<td>difference interval</td>
<td>Not applicable: GAMS provides analytic derivatives</td>
<td>1.5e-8</td>
</tr>
<tr>
<td>function precision</td>
<td>Relative accuracy with which the nonlinear functions are evaluated</td>
<td>3.00e-13</td>
</tr>
<tr>
<td>major iterations limit</td>
<td>Max number of major iterations</td>
<td>1000</td>
</tr>
<tr>
<td>major step limit</td>
<td>Limits the change in x during a linesearch</td>
<td>2.0</td>
</tr>
<tr>
<td>minor iterations limit</td>
<td>Max number of minor iterations between linearizations of nonlinear constraints</td>
<td>500</td>
</tr>
<tr>
<td>new superbasics limit</td>
<td>Limit on new superbasics when a QP subproblem is solved</td>
<td>99</td>
</tr>
<tr>
<td>nonderivative linesearch</td>
<td>Linesearch method (safeguarded quadratic interpolation) without use of derivatives</td>
<td></td>
</tr>
<tr>
<td>penalty parameter</td>
<td>Initial penalty parameter</td>
<td>0</td>
</tr>
<tr>
<td>proximal point method</td>
<td>Satisfies linear constraints near x0</td>
<td>1</td>
</tr>
<tr>
<td>reduced hessian dimension</td>
<td>Size of Hessian matrix</td>
<td>1</td>
</tr>
<tr>
<td>superbasics limit</td>
<td>Maximum number of superbasics</td>
<td>1</td>
</tr>
<tr>
<td>unbounded objective value</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e15</td>
</tr>
<tr>
<td>unbounded step size</td>
<td>Determines when a problem is called unbounded</td>
<td>1.0e18</td>
</tr>
<tr>
<td>violation limit</td>
<td>Limit on maximum constraint violation after the linesearch</td>
<td>10</td>
</tr>
<tr>
<td>warm start</td>
<td>Use advanced basis provided by GAMS</td>
<td></td>
</tr>
</tbody>
</table>
### 4.9 Hessian approximation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian frequency</td>
<td>How often the full Hessian is reset to the identity matrix</td>
<td>999999</td>
</tr>
<tr>
<td>hessian full memory</td>
<td>Approximate Hessian is treated as a dense matrix</td>
<td></td>
</tr>
<tr>
<td>hessian limited memory</td>
<td>Limited-memory procedure is used to update a diagonal Hessian approximation</td>
<td></td>
</tr>
<tr>
<td>hessian updates</td>
<td>How often the limited memory Hessian is reset</td>
<td>10</td>
</tr>
</tbody>
</table>

### 4.10 Frequencies

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>check frequency</td>
<td>Number of iterations between numerical accuracy check</td>
<td>60</td>
</tr>
<tr>
<td>expand frequency</td>
<td>Setting for anti-cycling mechanism</td>
<td>10000</td>
</tr>
<tr>
<td>factorization frequency</td>
<td>Number of iterations between basis factorizations</td>
<td>100 or 50</td>
</tr>
</tbody>
</table>

### 4.11 LUSOL options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU complete pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU density tolerance</td>
<td>When to use dense factorization</td>
<td>0.5</td>
</tr>
<tr>
<td>LU factor tolerance</td>
<td>Trade-off between stability and sparsity in basis factorization</td>
<td>3.99</td>
</tr>
<tr>
<td>LU partial pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU rook pivoting</td>
<td>LUSOL pivoting strategy</td>
<td></td>
</tr>
<tr>
<td>LU singularity tolerance</td>
<td>Protection against ill-conditioned basis matrices</td>
<td>3.2e-11</td>
</tr>
<tr>
<td>LU update tolerance</td>
<td>Trade-off between stability and sparsity in basis factorization</td>
<td>3.99</td>
</tr>
</tbody>
</table>

**central difference interval** *(real)*: Not applicable: GAMS provides analytic derivatives

When *Derivative level < 3* the central-difference interval \( r \) is used near an optimal solution to obtain more accurate (but more expensive) estimates of gradients. Twice as many function evaluations are required compared to forward differencing. The interval used for the \( j \)th variable is \( h_j = r(1 + |x_j|) \). The resulting derivative estimates should be accurate to \( O(r^2) \), unless the functions are badly scaled.

*(default = 6.0e-6)*

**check frequency** *(integer)*: Number of iterations between numerical accuracy check

Every \( i \)th minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution \( x \) satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form \( Ax - s = b \), where \( s \) is the set of slack variables. To perform the numerical test, the residual vector \( r = b - Ax + s \) is computed. If the largest component of \( r \) is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

**check frequency 1** is useful for debugging purposes, but otherwise this option should not be needed.

*(default = 60)*

**cold start** *(string)*: Ignore advanced basis and use CRASH procedure

Requests that the CRASH procedure be used to choose an initial basis.

**crash option** *(integer)*: Controls the basis crash algorithm

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the
constraint matrix \((A - I)\). The Crash option \(i\) determines which rows and columns of \(A\) are eligible initially, and how many times CRASH is called. Columns of \(-I\) are used to pad the basis where necessary.

If \(i \geq 1\), certain slacks on inequality rows are selected for the basis first. (If \(i \geq 2\), numerical values are used to exclude slacks that are close to a bound.) CRASH then makes several passes through the columns of \(A\), searching for a basis matrix that is essentially triangular. A column is assigned to "pivot" on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For remaining unassigned rows, slack variables are inserted to complete the basis.

(default = 0 or 3)

0 Initial basis will be a slack basis. The initial basis contains only slack variables: \(B = I\).

1 One phase CRASH. CRASH is called once, looking for a triangular basis in all rows and columns of the matrix \(A\).

2 Two phase CRASH. CRASH is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and CRASH is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).

3 Three phase CRASH. CRASH is called up to three times (if there are nonlinear constraints). The first two calls treat linear equalities and linear inequalities separately. As before, the last call treats nonlinear rows before the first major iteration.

**crash tolerance (real):** Allow crash procedure to ignore small elements in eligible columns

The Crash tolerance \(r\) allows the starting procedure CRASH to ignore certain small nonzeros in each column of \(A\). If \(a_{max}\) is the largest element in column \(j\), other nonzeros \(a_{ij}\) in the column are ignored if \(|a_{ij}| \leq a_{max} \times r\). (To be meaningful, \(r\) should be in the range \(0 \leq r < 1\).)

When \(r > 0.0\), the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of \(A\) and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first \(m\) columns of \(A\) are the matrix shown under LU factor tolerance; i.e., a tridiagonal matrix with entries \(-1, 2, -1\). To help CRASH choose all \(Sm\$\) columns for the initial basis, we would specify Crash tolerance \(r\) for some value of \(r > 0.5\).

(default = 0.1)

**derivative level (integer):** Specifies which derivatives are provided

The keyword Derivative level specifies which nonlinear function gradients are known analytically and will be supplied to SNOPT.

The value \(i = 3\) should be used whenever possible. It is the most reliable and will usually be the most efficient (default = 3)

0 Derivative level 0: Some components of the objective gradient are unknown and some of the constraint gradients are unknown.

1 Derivative level 1: The objective gradient is known, but some or all of the constraint gradients are unknown.

2 Derivative level 2: All constraint gradients are known, but some or all components of the objective gradient are unknown.

3 Derivative level 3: All objective and constraint gradients are known.

**derivative linesearch (string):** Linesearch method (safeguarded cubic interpolation) with use of derivatives

At each major iteration a linesearch is used to improve the merit function. A Derivative linesearch uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step \(\alpha_k\).
**difference interval (real):** Not applicable: GAMS provides analytic derivatives

This alters the interval $h_1$ that is used to estimate gradients by forward differences in the following circumstances:

- In the initial ("cheap") phase of verifying the problem derivatives.
- For verifying the problem derivatives.
- For estimating missing derivatives.

In all cases, a derivative with respect to $x_j$ is estimated by perturbing that component of $x$ to the value $x_j + h_1(1 + |x_j|)$, and then evaluating $f_0(x)$ or $f(x)$ at the perturbed point. The resulting gradient estimates should be accurate to $O(h_1)$ unless the functions are badly scaled. Judicious alteration of $h_1$ may sometimes lead to greater accuracy. This option has limited use in a GAMS environment as GAMS provides analytical gradients.

(default = $1.5 \times 10^{-8}$)

**elastic weight (real):** Used only during elastic mode

This parameter denoted by $\omega$ determines the initial weight $\gamma$ associated with problem NP$(\gamma)$. At any given major iteration $k$, elastic mode is started if the QP subproblem is infeasible, or the QP dual variables are larger in magnitude than $\omega(1 + \|g(x_k)\|_2)$, where $g$ is the objective gradient. In either case, the QP is re-solved in elastic mode with $\gamma = \omega(1 + \|g(x_k)\|_2)$.

Thereafter, $\gamma$ is increased (subject to a maximum allowable value) at any point that is optimal for problem NP$(\gamma)$, but not feasible for NP. After the $r$th increase, $\gamma = \omega 10^r (1 + \|g(x_{k1})\|_2)$, where $x_{k1}$ is the iterate at which $\gamma$ was first needed.

(default = $1.0 \times 10^4$)

**expand frequency (integer):** Setting for anti-cycling mechanism

This option is part of the EXPAND anti-cycling procedure [8] designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the Minor feasibility tolerance is $\delta$. Over a period of $i$ iterations, the tolerance actually used by SNOPT increases from $0.5\delta$ to $\delta$ (in steps of $0.5\delta/i$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

Increasing the expand frequency helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see pivot tolerance).

(default = 10000)

**factorization frequency (integer):** Number of iterations between basis factorizations

At most $k$ basis changes will occur between factorizations of the basis matrix.

- With linear programs, the basis factors are usually updated every iteration. The default $k$ is reasonable for typical problems. Smaller values (say $k = 75$ or $k = 50$) may be more efficient on problems that are rather dense or poorly scaled.
- When the problem is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the Check frequency) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of $k$ updates is reached.

Default: Factorization frequency 100 for linear programs and Factorization frequency 50 for nonlinear models.

(default = 100 or 50)

**feasible point (string):** Ignore objective function and find a feasible point
The keyword feasible point means "Ignore the objective function" while finding a feasible point for the linear and nonlinear constraints. It can be used to check that the nonlinear constraints are feasible.

Default: turned off.

**function precision (real):** Relative accuracy with which the nonlinear functions are evaluated

The relative function precision $\varepsilon_R$ is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as 1000.56789 for some relevant $x$ and if the first 6 significant digits are known to be correct, the appropriate value for $\varepsilon_R$ would be $1.0\times10^{-6}$.

(Ideally the functions $f_0(x)$ or $f_i(x)$ should have magnitude of order 1. If all functions are substantially less than 1 in magnitude, $\varepsilon_R$ should be the absolute precision. For example, if $f(x) = 1.23456789\times10^{-4}$ at some point and if the first 6 significant digits are known to be correct, the appropriate value for $\varepsilon_R$ would be $1.0\times10^{-10}$.)

- The default value of $\varepsilon_R$ is appropriate for simple analytic functions.
- In some cases the function values will be the result of extensive computation, possibly involving an iterative procedure that can provide rather few digits of precision at reasonable cost. Specifying an appropriate Function precision may lead to savings, by allowing the linesearch procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

(default $= 3.00\times10^{-13}$)

**hessian frequency (integer):** How often the full Hessian is reset to the identity matrix

If Hessian Full is selected and SIS BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

Hessian Full memory and Hessian frequency $= 20$ have a similar effect to Hessian Limited memory and Hessian updates $= 20$ (except that the latter retains the current diagonal during resets).

Default: Hessian frequency 99999999 (i.e. never).

(default $= 999999$)

**hessian full memory (string):** Approximate Hessian is treated as a dense matrix

This option selects the full storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

If Hessian Full memory is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables $n_1$ is not too large (say, less than 75). In this case, the storage requirement is fixed and one can expect Q-superlinear convergence to the solution.

Default: turned on when the number of nonlinear variables $n_1 \leq 75$.

**hessian limited memory (string):** Limited-memory procedure is used to update a diagonal Hessian approximation

This option selects the limited memory storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

Hessian Limited memory should be used on problems where the number of nonlinear variables $n_1$ is very large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation $H_{r}$ a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after $H_{r}$ has been reset to their diagonal.)

Default: turned on when the number of nonlinear variables $n_1 > 75$.

**hessian updates (integer):** How often the limited memory Hessian is reset

If Hessian Limited memory is selected and $i$ BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.
Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the more vectors stored, the greater the cost of each QP iteration. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., \(i = 5\)).

Default: Hessian updates 20 (only when limited memory storage model is used).

\(\text{(default = 10)}\)

**infinite bound (real):** Bounds larger than this number are considered Infinity

If \(r > 0\), \(r\) defines the "infinite" bound \(\infBnd\) in the definition of the problem constraints. Any upper bound greater than or equal to \(\infBnd\) will be regarded as plus infinity (and similarly for a lower bound less than or equal to \(-\infBnd\)). If \(r \leq 0\), the default value is used.

\(\text{(default = 1.0e20)}\)

**iterations limit (integer):** Minor iteration limit (ITERLIM)

This is the maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. This option overrides the GAMS iterlim options.

\(\text{(default = 1000)}\)

**linesearch tolerance (real):** Accuracy required for steplength

This controls the accuracy with which a steplength will be located along the direction of search each iteration. At the start of each linesearch a target directional derivative for the merit function is identified. This parameter determines the accuracy to which this target value is approximated.

- \(t\) must be a real value in the range \(0.0 \leq t \leq 1.0\).
- The default value \(t = 0.9\) requests just moderate accuracy in the linesearch.
- If the nonlinear functions are cheap to evaluate (this is usually the case for GAMS models), a more accurate search may be appropriate; try \(t = 0.1, 0.01\) or \(0.001\). The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. In the case of running under GAMS where all gradients are known, try \(t = 0.99\). (The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.)

Default: Linesearch tolerance 0.9.

\(\text{(default = 0.1)}\)

**LU complete pivoting (string):** LUSOL pivoting strategy

The LUSOL factorization implements a Markowitz-type search for pivots that locally minimize the fill-in subject to a threshold pivoting stability criterion. The rook and complete pivoting options are more expensive than partial pivoting but are more stable and better at revealing rank, as long as the LU factor tolerance is not too large (say \(t_1 < 2.0\)).

When numerical difficulties are encountered, SNOPT automatically reduces the \(LU\) tolerances toward 1.0 and switches (if necessary) to rook or complete pivoting before reverting to the default or specified options at the next refactorization. (With System information Yes, relevant messages are output to the listing file.)

**LU density tolerance (real):** When to use dense factorization

The density tolerance \(r_1\) is used during LUSOL’s basis factorization \(B = LU\). Columns of \(L\) and rows of \(U\) are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds \(r_1\), the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense \(LU\) procedure. Raising the density tolerance towards 1.0 may give slightly sparser \(LU\) factors, with a slight increase in factorization time.

Default: LU density tolerance 0.6 for most machines. This value corresponds to \(\epsilon^{2/3}\), where \(\epsilon\) is the relative machine precision.

See also LU singularity tolerance.

\(\text{(default = 0.5)}\)
LU factor tolerance (real): Trade-off between stability and sparsity in basis factorization

LU factor tolerance $r_1$
LU update tolerance $r_2$

These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy $r_1, r_2 \geq 1.0$. The matrix $L$ is a product of matrices of the form

$$
\begin{pmatrix}
1 & \mu & 1 \\
\end{pmatrix},
$$

where the multipliers $\mu$ satisfy $|\mu| \leq r_i$. Smaller values of $r_i$ favor stability, while larger values favor sparsity.

- For large and relatively dense problems, $r_1 = 5.0$ (say) may give a useful improvement in stability without impairing sparsity to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to reduce $r_1$ and/or $r_2$ in order to achieve stability. For example, if the columns of $A$ include a submatrix of the form

$$
\begin{pmatrix}
2 & -1 \\
-1 & 2 & -1 \\
\vdots & \ddots & \ddots \\
-1 & 2 & -1 \\
\end{pmatrix},
$$

both $r_1$ and $r_2$ should be in the range $1.0 \leq r_i < 2.0$.

Defaults for linear models: LU factor tolerance 100.0 and LU update tolerance 10.0. The defaults for nonlinear models are LU factor tolerance 3.99 and LU update tolerance 3.99.

See also LU_update_tolerance.
(default = 3.99)

LU partial pivoting (string): LUSOL pivoting strategy

See LU_complete_pivoting.

LU rook pivoting (string): LUSOL pivoting strategy

See LU_complete_pivoting.

LU singularity tolerance (real): Protection against ill-conditioned basis matrices

The singularity tolerance $r_2$ helps guard against ill-conditioned basis matrices. After $B$ is refactorized, the diagonal elements of $U$ are tested as follows: if $|U_{ij}| \leq r_2$ or $|U_{ij}| < r_2 \max_i |U_{ij}|$, the $j$th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart)

Default: LU singularity tolerance $3.2e-11$ for most machines. This value corresponds to $\epsilon^{2/3}$, where $\epsilon$ is the relative machine precision.

See also LU_density_tolerance.
(default = $3.2e-11$)

LU update tolerance (real): Trade-off between stability and sparsity in basis factorization

See LU_factor_tolerance.
(default = 3.99)

major feasibility tolerance (real): Specifies how accurately the nonlinear constraints should be satisfied

This specifies how accurately the nonlinear constraints should be satisfied. The default value of $1.0e-6$ is appropriate when the linear and nonlinear constraints contain data to about that accuracy.

Let $\text{rowerr}$ be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$
\text{rowerr} = \max_i \text{viol}_i/\|x\| \leq \epsilon_r,
$$

(8)
where \( \text{viol}_i \) is the violation of the \( i \)th nonlinear constraint (\( i = 1 : \text{nnCon} \), \( \text{nnCon} \) being the number of nonlinear constraints).

In the GAMS/SNOPT iteration log, \( \text{rowerr} \) appears as the quantity labeled "Feasibl". If some of the problem functions are known to be of low accuracy, a larger Major feasibility tolerance may be appropriate.

(default = 1.0e−6)

**major iterations limit (integer)**: Max number of major iterations

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints.

Default: Major iterations limit \( \max \{ 1000, m \} \).

(default = 1000)

**major optimality tolerance (real)**: Specifies the final accuracy of the dual variables

Major optimality tolerance \( \varepsilon_d \). This specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution \( (x, s, \pi) \) such that

\[
\max \text{Comp} = \max_j \text{Comp}_j / \| \pi \| \leq \varepsilon_d,
\]

where \( \text{Comp}_j \) is an estimate of the complementarity slackness for variable \( j \) (\( j = 1 : n + m \)). The values \( \text{Comp}_j \) are computed from the final QP solution using the reduced gradients \( d_j = g_j - \pi^T a_j \) (where \( g_j \) is the \( j \)th component of the objective gradient, \( a_j \) is the associated column of the constraint matrix \( (A - I) \) and \( \pi \) is the set of QP dual variables):

\[
\text{Comp}_j = \begin{cases} 
  d_j \min \{x_j - l_j, 1\} & \text{if } d_j \geq 0; \\
  -d_j \min \{u_j - x_j, 1\} & \text{if } d_j < 0.
\end{cases}
\]

In the GAMS/SNOPT iteration log, \( \text{maxComp} \) appears as the quantity labeled "Optimal".

(default = 1.0e−6)

**major print level (integer)**: Amount of information printed during optimization (listing file)

This controls the amount of output to the GAMS listing file each major iteration. This output is only visible if the \texttt{sysout} option is turned on (see \texttt{sysout} in Section \texttt{GAMS Options}). Major print level 1 gives normal output for linear and nonlinear problems, and Major print level 11 gives additional details of the Jacobian factorization that commences each major iteration. In general, the value being specified may be thought of as a binary number of the form

\[ \text{Major print level JFDXbs} \]

where each letter stands for a digit that is either 0 or 1 as follows:

- \( a \) single line that gives a summary of each major iteration. (This entry in JFDXbs is not strictly binary since the summary line is printed whenever JFDXbs \( \geq 1 \).
- \( b \) BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if JFDXbs \( \geq 10 \).
- \( x \) \( x_k \), the nonlinear variables involved in the objective function or the constraints.
- \( d \) \( \pi_k \), the dual variables for the nonlinear constraints.
- \( F \) \( F(x_k) \), the values of the nonlinear constraint functions.
- \( J \) \( J(x_k) \), the Jacobian.

To obtain output of any items JFDXbs, set the corresponding digit to 1, otherwise to 0.

If \( J=1 \), the Jacobian will be output column-wise at the start of each major iteration. Column \( j \) will be preceded by the value of the corresponding variable \( x_j \) and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if \( J=1 \), there is no reason to specify \( X=1 \) unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

\[ 3 \ 1.250000D+01 \ BS \ 1 \ 1.00000E+00 \ 4 \ 2.00000E+00 \]
which would mean that $x_3$ is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

**Major print level 0** suppresses most output, except for error messages.

(default = 1)

**major step limit (real):** Limits the change in $x$ during a linesearch

This parameter limits the change in $x$ during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

1. A linesearch determines a step $\alpha$ over the range $0 < \alpha \leq \beta$, where $\beta$ is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on $\$x$ if all the constraints are linear. Normally, the first steplength tried is $\alpha_1 = \min(1, \beta)$.

2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of $x$ can lead to floating-point overflow. The parameter $r$ is therefore used to define a limit $\beta = r(1 + ||x||)/||p||$ (where $p$ is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \beta, \beta)$.  

3. Wherever possible, upper and lower bounds on $x$ should be used to prevent evaluation of nonlinear functions at meaningless points. The **Major step limit** provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.

4. In cases where several local optima exist, specifying a small value for $r$ may help locate an optimum near the starting point.

(default = 2.0)

**minor feasibility tolerance (real):** Feasibility tolerance applied to all variables and linear constraints

Minor feasibility tolerance $t$. SNOPT tries to ensure that all variables eventually satisfy their upper and lower bounds to within the tolerance $t$. This includes slack variables. Hence, general linear constraints should also be satisfied to within $t$.

Feasibility with respect to nonlinear constraints is judged by the **Major feasibility tolerance** (not by $t$).

- If the bounds and linear constraints cannot be satisfied to within $t$, the problem is declared infeasible. Let $s_{inf}$ be the corresponding sum of infeasibilities. If $s_{inf}$ is quite small, it may be appropriate to raise $t$ by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

- Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem. For example, if $f(x) = \sqrt{x_1} + \log x_2$, it is essential to place lower bounds on both variables. If $t = 1.0e-6$, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep $x$ as far away from singularities as possible.)

- If $\text{Scale option} \geq 1$, feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful).

- In reality, SNOPT uses $t$ as a feasibility tolerance for satisfying the bounds on $x$ and $\$s$ in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. SNOPT is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See the Elastic options.

(default = 1.0e-6)

**minor iterations limit (integer):** Max number of minor iterations between linearizations of nonlinear constraints

Minor iterations limit $k$. If the number of minor iterations for the optimality phase of the QP subproblem exceeds $k$, then all nonbasic QP variables that have not yet moved are frozen at their current values and the reduced QP is solved to optimality. Note that more than $k$ minor iterations may be necessary to solve the reduced QP to optimality. These extra iterations are necessary to ensure that the terminated point gives a suitable direction for the linesearch. In the major iteration log, a *t* at the end of a line indicates that the corresponding QP was
artificially terminated using the limit $k$. Note that Iterations limit defines an independent absolute limit on the total number of minor iterations (summed over all QP subproblems).

(default = 500)

**minor print level (integer):** Amount of information printed during optimization (listing file)

This controls the amount of output to the GAMS listing file during solution of the QP subproblems. This option is only useful if the **sysout** option is turned on (see **sysout** in Section GAMS Options). The value of $k$ has the following effect:

- $0$ No minor iteration output except error messages.
- $\geq 1$ A single line of output each minor iteration (controlled by Print frequency).
- $\geq 10$ Basis factorization statistics generated during the periodic refactorization of the basis (see factorization_frequency). Statistics for the first factorization each major iteration are controlled by the Major print level.

(default = 1)

**new superbasics limit (integer):** Limit on new superbasics when a QP subproblem is solved

This option causes early termination of the QP subproblems if the number of free variables has increased significantly since the first feasible point. If the number of new superbasics is greater than new superbasics limit the nonbasic variables that have not yet moved are frozen and the resulting smaller QP is solved to optimality. In the major iteration log, a ”<tt>T<TT>” at the end of a line indicates that the QP was terminated early in this way.

(default = 99)

**nonderivative linesearch (string):** Linesearch method (safeguarded quadratic interpolation) without use of derivatives

A nonderivative linesearch can be slightly less robust on difficult problems, and it is recommended that the default be used if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nonderivative linesearch may give a significant decrease in computation time. In a GAMS environment derivative_linesearch (the default) is more appropriate.

**partial price (integer):** Number of segments in partial pricing strategy

Partial Price $i$. This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each ”pricing” operation (when a nonbasic variable is selected to become superbasic).

- When $i = 1$, all columns of the constraint matrix $(A - I)$ are searched.
- Otherwise, $A$ and $I$ are partitioned to give $i$ roughly equal segments $A_j, I_j$ ($j = 1$ to $i$). If the previous pricing search was successful on $A_j, I_j$, the next search begins on the segments $A_{j+1}, I_{j+1}$. (All subscripts here are modulo $i$.)
- If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments $A_{j+2}, I_{j+2}$, and so on.
- Partial price $t$ (or $t/2$ or $t/3$) may be appropriate for time-stage models having $t$ time periods.

Default: Partial price 10 for linear models and Partial price 1 for nonlinear models.

(default = 10)

**penalty parameter (real):** Initial penalty parameter

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function. This functions contains penalty parameters, which may be increased to ensure descent.

(default = 0)

**pivot tolerance (real):** Used to prevent columns entering the basis making it almost singular
During solution of QP subproblems, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.

- When \( x \) changes to \( x + \alpha p \) for some search direction \( p \), a "ratio test" is used to determine which component of \( x \) reaches an upper or lower bound first. The corresponding element of \( p \) is called the pivot element.
- Elements of \( p \) are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance \( r \).
- It is common for two or more variables to reach a bound at essentially the same time. In such cases, the Minor Feasibility tolerance (say \( t \)) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of \( StS \) should therefore not be specified.
- To a lesser extent, the Expand frequency (say \( f \)) also provides some freedom to maximize the pivot element. Excessively large values of \( f \) should therefore not be specified.

Default: Pivot tolerance 3.7e-11 on most machines. This corresponds to \( \varepsilon^{2/3} \) where \( \varepsilon \) is the machine precision.

(print frequency (integer): Number of iterations between each log line (listing file)

Synonym: log_frequency

When sysout is turned on (see sysout in Section GAMS Options) and Minor print level \( \geq 1 \), a line of the QP iteration log will be printed on the listing file every \( k \)th minor iteration.

Default: Print frequency 1.

(proximal point method (integer): Satisfies linear constraints near \( x_0 \)

\( i = 1 \) or \( 2 \) specifies minimization of \( \| x - x_0 \|_1 \) or \( \frac{1}{2} \| x - x_0 \|_2^2 \) when the starting point \( x_0 \) is changed to satisfy the linear constraints (where \( x_0 \) refers to nonlinear variables).

(default = 1)

1 one-norm Use \( \| x - x_0 \|_1 \).

2 two-norm Use \( \| x - x_0 \|_2^2 \).

(qpsolver (string): QP Solver

This specifies the method used to solve system (5) for the search directions in phase 2 of the QP subproblem.

- The Cholesky QP solver is the most robust, but may require a significant amount of computation if the number of superbasics is large.
- The quasi-Newton QP solver does not require the computation of the \( R \) at the start of each QP subproblem. It may be appropriate when the number of superbasics is large but relatively few major iterations are needed to reach a solution (e.g., if SNOPT is called with a Warm start).
- The conjugate-gradient QP solver is appropriate for problems with large numbers of degrees of freedom (say, more than 2000 superbasics).

(default = Cholesky)

Cholesky full Cholesky factor QPSolver Cholesky holds the full Cholesky factor \( R \) of the reduced Hessian \( Z^T H Z \). As the minor iterations proceed, the dimension of \( R \) changes with the number of superbasic variables. If the number of superbasic variables needs to increase beyond the value of Reduced Hessian dimension, the reduced Hessian cannot be stored and the solver switches to QPSolver CG. The Cholesky solver is reactivated if the number of superbasics stabilizes at a value less than Reduced Hessian dimension.

CG quasi-Newton method QPSolver CG uses an active-set method similar to QPSolver QN, but uses the conjugate-gradient method to solve all systems involving the reduced Hessian.

QN conjugate-gradient method QPSolver QN solves the QP using a quasi-Newton method similar to that of MINOS. In this case, \( R \) is the factor of a quasi-Newton approximate Hessian.
reduced hessian dimension (integer): Size of Hessian matrix

Synonym: hessian_dimension

This specifies that an \( i \times i \) triangular matrix \( R \) is to be available for use by the QPSolver Cholesky option (to define the reduced Hessian according to \( R^T R = Z^T H Z \)). The value of \( i \) affects when QPSolver CG is activated.

Default: Reduced Hessian dimension = \( \min\{2000, n_1 + 1\} \)
(default = 1)

scale option (integer): Scaling of linear/nonlinear variables

Three scale options are available (list below).

The listing file will only show these values if the sysout option is turned on (see sysout in Section GAMS Options).

See also scale_option and scale_tolerance.

(default = 1)

0 No scaling This is recommended if it is known that \( x \) and the constraint matrix (and Jacobian) never have very large elements (say, larger than 100).

1 Scale linear variables Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer [5]). This will sometimes improve the performance of the solution procedures.

2 Scale linear + nonlinear variables All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of \( (A - I) \) that are fixed or have positive lower bounds or negative upper bounds.

If nonlinear constraints are present, the scales depend on the Jacobian at the first point that satisfies the linear constraints. Scale option 2 should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

scale print (string): Print scaling factors (listing file)

Scale print causes the row-scales \( r(i) \) and column-scales \( c(j) \) to be printed. The scaled matrix coefficients are \( \bar{a}_{ij} = a_{ij}c(j)/r(i) \), and the scaled bounds on the variables and slacks are \( \bar{l}_j = l_j/c(j) \), \( \bar{u}_j = u_j/c(j) \), where \( c(j) \equiv r(j - n) \) if \( j > n \).

The listing file will only show these values if the sysout option is turned on (see sysout in Section GAMS Options).

See also scale_option and scale_tolerance.

scale tolerance (real): Scale tolerance

Scale tolerance \( t \) affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

\[
\rho_j = \max_i |a_{ij}|/\min_i |a_{ij}| \quad (a_{ij} \neq 0).
\]

If \( \max_j \rho_j \) is less than \( r \) times its previous value, another scaling pass is performed to adjust the row and column scales. Raising \( r \) from 0.9 to 0.99 (say) usually increases the number of scaling passes through \( A \). At most 10 passes are made.

The listing file will only show these values if the sysout option is turned on (see sysout in Section GAMS Options).

See also scale_option and scale_print.

(default = 0.9)

solution (string): Prints SNOPT solution (listing file)

This option causes the SNOPT solution file to be printed to the GAMS listing file. It is only visible if the sysout option is turned on (see sysout in Section GAMS Options).
Default: turned off.

扭 off printing of solution

YES Turn on printing of solution

**start constraint check (integer):** Can be used to reduce the range of finite-difference checks

If $\text{Verify level} > 0$, these options may be used to abbreviate the verification of individual derivative elements.

This option is most useful when not running under a GAMS environment.

(default = 1)

**start objective check (integer):** Can be used to reduce the range of finite-difference checks

If $\text{Verify level} > 0$, these options may be used to abbreviate the verification of individual derivative elements.

This option is most useful when not running under a GAMS environment.

(default = 1)

**stop constraint check (integer):** Can be used to reduce the range of finite-difference checks

If $\text{Verify level} > 0$, these options may be used to abbreviate the verification of individual derivative elements.

This option is most useful when not running under a GAMS environment.

(default = MAXINT)

**stop objective check (integer):** Can be used to reduce the range of finite-difference checks

If $\text{Verify level} > 0$, these options may be used to abbreviate the verification of individual derivative elements.

This option is most useful when not running under a GAMS environment.

(default = MAXINT)

**summary frequency (integer):** Number of iterations between each log line (log file)

If $\text{Minor print level} > 0$, a line of the QP iteration log is output every $k$th minor iteration.

(default = 100)

**superbasics limit (integer):** Maximum number of superbasics

This places a limit on the storage allocated for superbasic variables. Ideally, $i$ should be set slightly larger than the "number of degrees of freedom" expected at an optimal solution.

For linear programs, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is no more than $\text{sm}$, the number of general constraints.) The default value of $i$ is therefore 1.

For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables".

Normally, $i$ need not be greater than $n_1 + 1$, where $n_1$ is the number of nonlinear variables. For many problems, $i$ may be considerably smaller than $n_1$. This will save storage if $n_1$ is very large.

(default = 1)

**suppress parameters (string):** Suppress printing of parameters (listing file)

Normally SNOPT prints the option file as it is being read, and the prints a complete list of the available keywords and their final values. The Suppress Parameters option tells SNOPT not to print the full list.

**system information (string):** Provides additional information on the progress of the iterations (listing file)

The Yes option provides additional information on the progress of the iterations, including Basis Repair details when ill-conditioned bases are encountered and the LU factorization parameters are strengthened. parameters are strengthened.

(default = NO)

扭 off additional printing of information on progress of algorithm

YES Turn on additional printing of information on progress of algorithm
**timing level** (integer): Amount of timing information (listing file)

Amount of timing information written to the listing file. This is visible in combination with the GAMS setting `OPTION SYSOUT=on;`.

(default = 3)

**unbounded objective value** (real): Determines when a problem is called unbounded

Unbounded objective value $f_{\text{max}}$. This parameter is intended to detect unboundedness in nonlinear problems. (They may not achieve that purpose!) During a line search, $f_0$ is evaluated at points of the form $x + \alpha p$, where $x$ and $p$ are fixed and $\alpha$ varies. If $|f_0|$ exceeds $f_{\text{max}}$ or $\alpha$ exceeds $\alpha_{\text{max}}$, iterations are terminated with the exit message **Problem is unbounded (or badly scaled)**.

Unboundedness in $x$ is best avoided by placing finite upper and lower bounds on the variables.

In a GAMS environment no floating-point overflow errors should occur when singularities are present during the evaluation of $f(x + \alpha p)$ before the test can be made.

See also **unbounded step size**.

(default = 1.0e15)

**unbounded step size** (real): Determines when a problem is called unbounded

Unbounded step size $\alpha_{\text{max}}$. This parameter is intended to detect unboundedness in nonlinear problems. (They may not achieve that purpose!) During a line search, $f_0$ is evaluated at points of the form $x + \alpha p$, where $x$ and $p$ are fixed and $\alpha$ varies. If $|f_0|$ exceeds $f_{\text{max}}$ or $\alpha$ exceeds $\alpha_{\text{max}}$, iterations are terminated with the exit message **Problem is unbounded (or badly scaled)**.

Unboundedness in $x$ is best avoided by placing finite upper and lower bounds on the variables.

In a GAMS environment no floating-point overflow errors should occur when singularities are present during the evaluation of $f(x + \alpha p)$ before the test can be made.

See also **unbounded objective value**.

(default = 1.0e18)

**verify level** (integer): Finite-difference checks on the derivatives

This option refers to finite-difference checks on the derivatives computed by the user-provided routines. Derivatives are checked at the first point that satisfies all bounds and linear constraints. Linear constraints.

This option has limited use in a GAMS environment.

(default = 0)

0 Cheap test
1 Check individual gradients
2 Check individual columns of the Jacobian
3 Combines verify level 1 and 2
-1 Derivative checking is disabled

**violation limit** (integer): Limit on maximum constraint violation after the linesearch

This keyword defines an absolute limit on the magnitude of the maximum constraint violation after the line search. On completion of the line search, the new iterate $x_{k+1}$ satisfies the condition

$$v_i(x_{k+1}) \leq \tau \max \{1, v_i(x_0)\},$$

where $x_0$ is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the $i$th nonlinear constraint violation $v_i(x) = \max(0, l_i - F_i(x), F_i(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an expanded feasible region whose size depends on the magnitude of $\tau$. This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then $\tau$ may be any large positive value.
SNOPT 513

(default = 10)

**warm start (string):** Use advanced basis provided by GAMS

Use an advanced basis provided by GAMS.

5 The SNOPT log

When GAMS/SNOPT solves a linearly constrained problem the following log is visible on the screen:

--- Job chem Start 04/29/13 19:40:09 LEX-LEG 24.0.2 x86_64/Linux
GAMS Rev 240 Copyright (C) 1987-2013 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC  G871201/0000CA-ANY
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC0000
--- Starting compilation
--- chem.gms(49) 3 Mb
--- Starting execution: elapsed 0:00:00.002
--- chem.gms(45) 4 Mb
--- Generating NLP model mixer
--- chem.gms(49) 6 Mb
--- 5 rows 12 columns 37 non-zeros
--- 72 nl-code 11 nl-non-zeros
--- chem.gms(49) 4 Mb
--- Executing SNOPT: elapsed 0:00:00.003

SNOPT Feb 14, 2013 24.0.2 LEX 38380.38394 LEG x86_64/Linux

GAMS/SNOPT, Large Scale Nonlinear SQP Solver
S N O P T 7.2-12 (May 2011)
P. E. Gill, UC San Diego
W. Murray and M. A. Saunders, Stanford University
Reading Rows...
Reading Columns...
Reading Instructions...

Work space estimate computed by solver -- 0.20 MB

<table>
<thead>
<tr>
<th>Major</th>
<th>Minor</th>
<th>Step</th>
<th>nObj</th>
<th>Objective</th>
<th>Optimal</th>
<th>nS PD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>0.0E+00</td>
<td>1</td>
<td>3.292476E+01</td>
<td>1.8E-01</td>
<td>5 TF r</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4.1E+00</td>
<td>3</td>
<td>4.131132E+01</td>
<td>1.6E-01</td>
<td>6 TF N r</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.4E+00</td>
<td>5</td>
<td>4.277073E+01</td>
<td>1.2E-01</td>
<td>6 TF s</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2.3E+00</td>
<td>7</td>
<td>4.611470E+01</td>
<td>1.3E-01</td>
<td>5 TF</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>9.8E-01</td>
<td>10</td>
<td>4.759489E+01</td>
<td>8.0E-02</td>
<td>4 TF</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1.9E-01</td>
<td>14</td>
<td>4.763815E+01</td>
<td>2.3E-02</td>
<td>4 TF</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5.8E-01</td>
<td>16</td>
<td>4.767446E+01</td>
<td>2.9E-02</td>
<td>5 TF</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5.0E-01</td>
<td>18</td>
<td>4.768810E+01</td>
<td>1.4E-02</td>
<td>5 TF</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8.0E-01</td>
<td>20</td>
<td>4.770052E+01</td>
<td>9.0E-03</td>
<td>6 TF</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>8.4E-02</td>
<td>23</td>
<td>4.770088E+01</td>
<td>8.4E-03</td>
<td>6 TF</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1.0E+00</td>
<td>24</td>
<td>4.770589E+01</td>
<td>2.5E-03</td>
<td>6 TF</td>
</tr>
</tbody>
</table>

Major Minor | Step | nObj | Objective | Optimal | nS PD |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>9.3E-01</td>
<td>26</td>
<td>4.770648E+01</td>
<td>1.4E-03</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1.1E+00</td>
<td>28</td>
<td>4.770651E+01</td>
<td>6.2E-04</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1.4E+00</td>
<td>30</td>
<td>4.770651E+01</td>
<td>9.6E-05</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1.1E+00</td>
<td>32</td>
<td>4.770651E+01</td>
<td>4.1E-06</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>9.5E-01</td>
<td>34</td>
<td>4.770651E+01</td>
<td>1.0E-07</td>
</tr>
</tbody>
</table>

EXIT - Optimal Solution found, objective: -47.70651
--- Restarting execution
--- chem.gms(49) 2 Mb
--- Reading solution for model mixer
*** Status: Normal completion
--- Job chem.gms Stop 04/29/13 19:40:09 elapsed 0:00:00.035

For a nonlinearly constrained problem, the log is somewhat different:

--- Job chenery.gms Start 04/29/13 19:41:12 LEX-LEG 24.0.2 x86_64/Linux
GAMS Rev 240 Copyright (C) 1987-2013 GAMS Development. All rights reserved Licensee: GAMS Development Corporation, Washington, DC G871201/000OCA-ANY
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC0000
--- Starting compilation
--- chenery.gms(241) 3 Mb
--- Starting execution: elapsed 0:00:00.002
--- chenery.gms(224) 4 Mb
--- Generating NLP model chenrad
--- chenery.gms(227) 6 Mb
--- 39 rows 44 columns 133 non-zeroes
--- 194 nl-code 56 nl-non-zeroes
--- chenery.gms(227) 4 Mb
--- Executing SNOPT: elapsed 0:00:00.005

SNOPT Feb 14, 2013 24.0.2 LEX 38380.38394 LEG x86_64/Linux

GAMS/SNOPT, Large Scale Nonlinear SQP Solver
S N O P T 7.2-12 (May 2011)
P. E. Gill, UC San Diego
W. Murray and M. A. Saunders, Stanford University

Reading Rows...
Reading Columns...
Reading Instructions...

Work space estimate computed by solver -- 0.26 MB

<table>
<thead>
<tr>
<th>Itn</th>
<th>Major</th>
<th>Minor</th>
<th>Step</th>
<th>nCon</th>
<th>Merit</th>
<th>Feasibl</th>
<th>Optimal</th>
<th>nS</th>
<th>Penalty</th>
<th>PD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>29</td>
<td>0.0E+00</td>
<td>1</td>
<td>6.000000E+02</td>
<td>1.4E+00</td>
<td>9.0E-03</td>
<td>4</td>
<td>0.0E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5.8E-03</td>
<td>2</td>
<td>-9.659720E+06</td>
<td>1.4E+00</td>
<td>1.0E+00</td>
<td>4</td>
<td>7.6E-01</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6.3E-03</td>
<td>4</td>
<td>-5.124694E+05</td>
<td>1.4E+00</td>
<td>1.0E-01</td>
<td>4</td>
<td>7.6E-01</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>1.0E+00</td>
<td>6</td>
<td>4.088952E+04</td>
<td>9.8E-01</td>
<td>8.4E-02</td>
<td>5</td>
<td>7.6E-01</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1.0E+00</td>
<td>8</td>
<td>-1.379500E+04</td>
<td>4.9E-01</td>
<td>1.4E-01</td>
<td>5</td>
<td>1.7E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1.0E+00</td>
<td>9</td>
<td>-1.506650E+03</td>
<td>1.3E-01</td>
<td>7.3E-01</td>
<td>5</td>
<td>1.7E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1.0E+00</td>
<td>11</td>
<td>4.635672E+02</td>
<td>2.2E-02</td>
<td>4.1E-01</td>
<td>3</td>
<td>1.7E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>1.5E-01</td>
<td>19</td>
<td>4.561587E+02</td>
<td>2.4E-02</td>
<td>5.9E-01</td>
<td>2</td>
<td>1.7E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1.4E-01</td>
<td>26</td>
<td>5.306161E+02</td>
<td>2.4E-02</td>
<td>1.3E+00</td>
<td>1</td>
<td>1.7E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1.4E-01</td>
<td>33</td>
<td>5.665326E+02</td>
<td>2.5E-02</td>
<td>1.4E+00</td>
<td>1</td>
<td>1.7E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>1.4E-01</td>
<td>40</td>
<td>6.039374E+02</td>
<td>2.5E-02</td>
<td>5.8E-01</td>
<td>1</td>
<td>1.7E+00</td>
<td>FF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- Restarting execution
--- chem.gms(49) 2 Mb
--- Reading solution for model mixer
*** Status: Normal completion
--- Job chem.gms Stop 04/29/13 19:40:09 elapsed 0:00:00.035
EXIT - Optimal Solution found, objective: 1058.920

--- Restarting execution
--- chenery.gms(227) 2 Mb
--- Reading solution for model chenrad
--- chenery.gms(241) 3 Mb
*** Status: Normal completion
--- Job chenery.gms Stop 04/29/13 19:41:12 elapsed 0:00:00.076

GAMS prints the number of equations, variables and non-zero elements of the model it generated. This gives an indication of the size of the model. SNOPT then says how much memory it allocated to solve the model, based on an estimate. If the user had specified a different amount using the work option or the workspace model suffix, there would be a message like

Work space estimate computed by solver -- 0.26 MB
Work space value given by user -- 0.76 MB

The SNOPT log shows the following columns:

**Major** The current major iteration number.

**Minor** is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, Minor will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section Description of the method).
**Step**  The step length $\alpha$ taken along the current search direction $p$. The variables $x$ have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

**nObj**  The number of times the nonlinear objective function has been evaluated. nObj is printed as a guide to the amount of work required for the linesearch.

**nCon**  The number of times SNOPT evaluated the nonlinear constraint functions.

**Merit**  is the value of the augmented Lagrangian merit function (6).

This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section Description of the method). As the solution is approached, Merit will converge to the value of the objective at the solution.

In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.

If the constraints are linear, this item is labeled **Objective**, the value of the objective function. It will decrease monotonically to its optimal value.

**Feasibl**  is the value of rowerr, the maximum component of the scaled nonlinear constraint residual ($\epsilon$). The solution is regarded as acceptably feasible if Feasibl is less than the Major feasibility tolerance.

If the constraints are linear, all iterates are feasible and this entry is not printed.

**Optimal**  is the value of maxgap, the maximum complementarity gap ($\epsilon$).

It is an estimate of the degree of nonoptimality of the reduced costs. Both Feasibl and Optimal are small in the neighborhood of a solution.

**nS**  The current number of superbasic variables.

**Penalty**  is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if the constraints are linear).

**PD**  is a two-letter indication of the status of the convergence tests involving primal and dual feasibility of the iterates (see (and () in the description of Major feasibility tolerance and Major optimality tolerance). Each letter is T if the test is satisfied, and F otherwise.

If either of the indicators is F when SNOPT terminates with 0 EXIT -- optimal solution found, the user should check the solution carefully.

The summary line may include additional code characters that indicate what happened during the course of the iteration.

- **c** Central differences have been used to compute the unknown components of the objective and constraint gradients. This should not happen in a GAMS environment.
- **d** During the linesearch it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of Violation limit.
- **l** The norm-wise change in the variables was limited by the value of the Major step limit. If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of Major step limit.
- **i** If SNOPT is not in elastic mode, an "i" signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem NP($\gamma$).
- **M** An extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.
This is the same as "M" except that it was also necessary to modify the update to include an augmented Lagrangian term.

R The approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the Hessian frequency and Hessian updates keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

r The approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

s A self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.

S This is the same as a "s" except that it was necessary to modify the self-scaled update to maintain positive definiteness.

n No positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.

t The minor iterations were terminated at the Minor iteration limit.

u The QP subproblem was unbounded.

A weak solution of the QP subproblem was found.

Finally SNOPT prints an exit message. See Section EXIT conditions.

5.1 EXIT conditions

When the solution procedure terminates, an EXIT -- message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

EXIT - Optimal Solution found, objective: xx.xx

The final point seems to be a solution of NP. This means that x is feasible (it satisfies the constraints to the accuracy requested by the Feasibility tolerance), the reduced gradient is negligible, the reduced costs are optimal, and R is nonsingular. In all cases, some caution should be exercised. For example, if the objective value is much better than expected, SNOPT may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are minimizing the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the same objective value, particularly if the objective is largely linear.) Conversely, if we are maximizing a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Our advice is always to specify a starting point that is as good an estimate as possible, and to include reasonable upper and lower bounds on all variables, in order to confine the solution to the specific region of interest. We expect modelers to know something about their problem, and to make use of that knowledge as they themselves know best.

One other caution about "Optimal solution" s. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. Max Primal infeas refers to the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller Minor feasibility tolerance (say 10 times smaller) and perhaps Scale option 0.

Similarly, Max Dual infeas indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

Max Dual infeas/Norm of pi = 10^{-d},
then the objective function would probably change in the $d$th significant digit if optimization could be continued.

If $d$ seems too large, consider restarting with smaller Major and Minor optimality tolerances.

Finally, Nonlinear constraint violn shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller Major feasibility tolerance.

EXIT -- Feasible point found, objective: xx.xx
From option Feasible point only.

EXIT -- Requested accuracy could not be achieved, objective:
If the requested accuracy could not be achieved, a feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but SNOPT is within $10^{-2}$ of satisfying the Major optimality tolerance. Check that the Major optimality tolerance is not too small.

EXIT -- The problem is infeasible (infeasible linear constraints)
EXIT -- The problem is infeasible (infeasible linear equalities)
When the constraints are linear, the output messages are based on a relatively reliable indicator of infeasibility. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints $Ax - s = 0$, there is apparently no point that satisfies the bounds on $x$ and $s$. Violations as small as the Minor feasibility tolerance are ignored, but at least one component of $x$ or $s$ violates a bound by more than the tolerance.

EXIT -- Nonlinear infeasibilities minimized
EXIT -- Infeasibilities minimized
When nonlinear constraints are present, infeasibility is much harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, SNOPT is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters so-called "nonlinear elastic" mode. The subproblem includes the original QP objective and the sum of the infeasibilities—suitably weighted using the 'Elastic weight" parameter. In elastic mode, the nonlinear rows are made "elastic"—i.e., they are allowed to violate their specified bounds. Variables subject to elastic bounds are known as elastic variables. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, SNOPT will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, SNOPT would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though SNOPT locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

EXIT -- Unbounded objective
EXIT -- Unbounded: Constraint violation limit reached
For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. Adding a bound on the objective will allow SNOPT to find a solution, and inspection of this solution will show the variables that can become too large to missing restrictions.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the Scale option.

For nonlinear problems, SNOPT monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the Unbounded parameters—see Section Options), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may
be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the Violation limit.

EXIT -- User Interrupt
The user pressed Ctrl-C or the Interrupt button in the Windows IDE.

EXIT -- Resource Interrupt
A time limit was hit. Increase the GAMS reslim option.

EXIT -- Too many iterations (exceeding ITERLIM)
An iteration limit was reached. Most often this is cured by increasing the GAMS iterlim option. If an SNOPT option file was used, also the Iterations limit may have been set too small.

Check the iteration log to be sure that progress was being made. If so, repeat the run with higher limits. If not, consider specifying new initial values for some of the nonlinear variables.

EXIT -- Major iteration limit reached
This indicates SNOPT was running out the limit on major iterations. This can be changed using the Major iterations limit.

EXIT -- The superbasics limit is too small
The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already as many superbasics as allowed (and no room for any more).

When increasing the superbasics limit also note that it is needed to increase the amount of available memory. This can be done with the GAMS m.workspace and m.workfactor model suffices.

EXIT -- Current point cannot be improved
The algorithm could not find a better solution although optimality was not achieved within the optimality tolerance. Possibly scaling can lead to better function values and derivatives. Raising the optimality tolerance will probably make this message go away. Try better scaling, better bounds or a better starting point.

EXIT -- Singular basis
The first factorization attempt found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix $U$ were deemed too small.) The associated variables were replaced by slacks and the modified basis refactorized, but singularity persisted. Try better scaling, better bounds or a better starting point.

EXIT -- Cannot satisfy the general constraints
The basic variables $x_B$ have been recomputed, given the present values of the superbasic and nonbasic variables. A step of "iterative refinement" has also been applied to increase the accuracy of $x_B$, but a row check has revealed that the resulting solution does not satisfy the QP constraints $Ax - s = b$ sufficiently well. Try better scaling, better bounds or a better starting point.

EXIT -- Ill-conditioned null-space basis
During computation of the reduced Hessian $Z^T H Z$, some column(s) of $Z$ continued to contain very large values. Try better scaling, better bounds or a better starting point.

EXIT -- Incorrect objective derivatives
EXIT -- Incorrect constraint derivatives
The derivatives are not deemed to be correct. This message should not occur using a GAMS model without external functions.

EXIT -- Undefined function at the initial point
EXIT -- Undefined function at the first feasible point
SNOPT was unable to proceed because the functions are undefined at the initial point or the first feasible point. Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

EXIT -- Unable to proceed into undefined region
Repeated attempts to move into a region where the functions are not defined resulted in the change in variables being unacceptably small. At the final point, it appears that the only way to decrease the merit function is to move into a region where the problem functions are not defined.

Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

EXIT -- Function evaluation error limit
The domain error limit was reached. Increase the GAMS domlim option, or better add better bounds (or linear equations) such that functions and derivatives can be evaluated.

EXIT -- Terminated during objective evaluation
EXIT -- Terminated during constraint evaluation
EXIT -- Terminated from monitor routine
These messages indicate troubles evaluating the non-linear functions or derivatives. Usually these errors show a "Function evaluation error limit" message.

6 Listing file messages

The listing file .lst file) also contains feedback on how the SNOPT solver performed on a particular model. For the chenery.gms model, the solve summary looks like the following:

SOLVE SUMMARY

MODEL chenrad OBJECTIVE td
TYPE NLP DIRECTION MAXIMIZE
SOLVER SNOPT FROM LINE 227

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 2 LOCALLY OPTIMAL
**** OBJECTIVE VALUE 1058.9199

RESOURCE USAGE, LIMIT 0.063 1000.000
ITERATION COUNT, LIMIT 179 10000
EVALUATION ERRORS 0 0

SNOPT Feb 14, 2013 24.0.2 LEX 38380.38394 LEG x86_64/Linux

Table 1 : Model status values

<table>
<thead>
<tr>
<th>Model status</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 OPTIMAL</td>
<td>Applies only to linear models.</td>
</tr>
</tbody>
</table>
2 LOCALLY OPTIMAL  | A local optimum in an NLP was found. It may or may not be a global optimum.
3 UNBOUNDED         | For LP’s this message is reliable. A badly scaled NLP can also cause this message to appear.
4 INFEASIBLE        | Applies to LP’s: the model is infeasible.
5 LOCALLY INFEASIBLE| Applies to NLP’s: Given the starting point, no feasible solution could be found although feasible points may exist.
6 INTERMEDIATE INFEASIBLE | The search was stopped (e.g., because of an iteration or time limit) and the current point violates some constraints or bounds.
7 FEASIBLE SOLUTION | The search was stopped (e.g., because of an iteration or time limit) and the current point is feasible but violates the optimality conditions.
8 INTEGER SOLUTION  | Does not apply to SNOPT.
9 INTERMEDIATE NON–INTEGER | Does not apply to SNOPT.
10 INTEGER INFEASIBLE | Does not apply to SNOPT.
ERROR UKNOWN        | Check listing file for error messages.
ERROR NO SOLUTION    | Check listing file for error messages.

**Table 2**: Solver status value

<table>
<thead>
<tr>
<th>Solver status</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 NORMAL COMPLETION</td>
<td>SNOPT completed the optimization task.</td>
</tr>
<tr>
<td>2 ITERATION INTERRUPT</td>
<td>Iteration limit was hit. Increase the <strong>iterlim</strong> option (see <strong>GAMS iterlim</strong> Section <strong>GAMS Options</strong>).</td>
</tr>
<tr>
<td>3 RESOURCE INTERRUPT</td>
<td>Time limit was hit. Increase the <strong>reslim</strong> option (see <strong>GAMS reslim</strong> in Section <strong>GAMS Options</strong>).</td>
</tr>
<tr>
<td>4 TERMINATED BY SOLVER</td>
<td>Check the listing file.</td>
</tr>
<tr>
<td>5 EVALUATION ERROR LIMIT</td>
<td><strong>domlim</strong> error limit was exceeded (see <strong>GAMS domlim</strong> in Section <strong>GAMS Options</strong>).</td>
</tr>
<tr>
<td>6 UNKNOWN ERROR</td>
<td>Check the listing file for error messages.</td>
</tr>
<tr>
<td>ERROR SETUP FAILURE</td>
<td>Id.</td>
</tr>
<tr>
<td>ERROR SOLVER FAILURE</td>
<td>Id.</td>
</tr>
<tr>
<td>ERROR INTERNAL SOLVER FAILURE</td>
<td>Id.</td>
</tr>
<tr>
<td>ERROR SYSTEM FAILURE</td>
<td>Id.</td>
</tr>
</tbody>
</table>

The solver completed normally at a local (or possibly global) optimum. A complete list of possible solver status and model status values is in Tables Table 1 and Table 2.

The resource usage (time used), iteration count and evaluation errors during nonlinear function and gradient evaluation are all within their limits. These limits can be increased by the option **reslim**, **iterlim** and **domlim** (see Section **GAMS Options**).

The possible EXIT messages are listed in Section **EXIT conditions**.

**Bibliography**


1 Introduction

Sulum Optimization Tools includes a high performance optimizer for Linear and Mixed-Integer Linear Programming Problems (LP and MIP).

The LP optimizer is based on a very efficient simplex code, which uses state of the art implementation and algorithmic techniques. The key features of the LP optimizer can be summarized as:

- Strong LP presolve to reduce problem size.
- Highly optimized sparse vector and matrix implementation.
- Fast and stable LU factorization and sparsity exploiting solves.
- Numerical stabilizing techniques, which include auto switching to using quad precision if deemed needed.
- Multiple pricing options i.e steepest edge, approximate steepest edge and partial pricing.
- Advanced crash of initial basis.
- Auto dualizer switch to solving the dual problem if it is smaller than the primal problem.
- Fast reoptimize from a previous found solution.

The MIP optimizer is an advanced implementation of a branch and cut method, with many performance enhancements added. In the coming months more performance features will be added. The key features of the MIP optimizer can be highlighted as:

- Advanced branching strategies.
- Strong cutting plane techniques.
- Fast and stable branch and bound tree management.
- Advanced presolve module to further reduce problem size.
- Auto Benders decomposition to solve subproblems more efficiently.
- Fast reoptimize from a previous found solution.
- Advanced MIP presolve to reduce the problem size and provide a better formulation for the optimizer.
- Tight integration with the Sulum LP optimizer to efficiently solve LPs in node and during heuristics.
- Various branching and node selection methods from inexpensive to more expensive computational schemes.
- Cutting plane generation and filtering if deemed necessary.
- Heuristics to either find an initial solution or improve the current incumbent, which includes rounding, diving, and sub-mipping heuristics types.
- Improvements to ensure numerical stability.

While numerous solving options are available, Sulum automatically calculates and sets most options at the best values for specific problems. All Sulum options available through GAMS/Sulum are summarized at the end of this chapter.

2 How to Run a Model with Sulum

The following statement can be used inside your GAMS program to specify using Sulum

```gams
Option LP = Sulum; { or MIP or RMIP }
```

The above statement should appear before the `solve` statement. If Sulum was specified as the default solver during GAMS installation the above statement is not necessary.

3 Overview of GAMS/Sulum

Sulum can solve LPs using a simplex algorithm and MIPs using a branch and cut algorithm.

The Sulum presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Sulum can, in order to get better diagnostic information, rerun the problem with presolve turned off. The rerun without presolve is controlled by the option `rerun`. In default mode only problems that are small (i.e. demo sized) will be rerun.

For an LP, Sulum can either presolve a model or start from an advanced basis or primal/dual solution pair. Often the solve from scratch of a presolved model outperforms a solve from an unpresolved model started from an advanced basis/solution. It is impossible to determine a priori if presolve or starting from a given advanced basis/solution without presolve will be faster. By default, GAMS/Sulum will automatically use an advanced basis or solution from a previous solve statement. The GAMS BRatio option can be used to specify when not to use an advanced basis/solution. The GAMS/Sulum option `usebasis` can be used to ignore or force a basis/solution passed on by GAMS (it overrides BRatio). In case of multiple solves in a row and slow performance of the second and subsequent solves, the user is advised to set the GAMS BRatio option to 1.

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear programs. Many relatively small integer programming models take enormous amounts of time to solve. For problems with integer variables, Sulum uses a branch and cut algorithm which solves a series of LP subproblems. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

4 GAMS Options

The following GAMS options are used by GAMS/Sulum:

- `Option BRatio = x;`
Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Sulum not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed.

- **Option IterLim = n;**
  Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS. For MIP problems, if the number of the cumulative simplex iterations exceeds the limit, Sulum will terminate.

- **Option NodLim= x;**
  Maximum number of nodes to process for a MIP problem. This GAMS option is overridden by the GAMS/Sulum option `mipmaxnodes`.

- **Option OptCR = x;**
  Relative optimality criterion for a MIP problem. Notice that Sulum uses a different definition than GAMS normally uses. The OptCR option asks Sulum to stop when
  \[|BP - BF| < |BF| \ast OptCR\]
  where \(BF\) is the objective function value of the current best integer solution while $BP$ is the best possible integer solution. The GAMS definition is:
  \[|BP - BF| < |BP| \ast OptCR\]

- **Option ResLim = x;**
  Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. Sulum measures time in wall time on all platforms. Some other GAMS solvers measure time in CPU time on some Unix systems. This GAMS option is overridden by the GAMS/Sulum option `opttimelimit`.

- **Option SysOut = On;**
  Will echo Sulum messages to the GAMS listing file. This option may be useful in case of a solver failure.

- **ModelName.Cutoff= x;**
  Cutoff value. The algorithm terminates when it is proven that the optimal value is worse than the cutoff value. This GAMS option is overridden by the GAMS/Sulum options `simobjupcut` (for minimization problems) or `simobjlocut` (for maximization problems).
  When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm. This GAMS option is overridden by the GAMS/Sulum option `mipcutoff`.

- **ModelName.OptFile=1;**
  Instructs GAMS/Sulum to read the option file. The name of the option file is `sulum.opt`.

- **ModelName.PriorOpt= 1;**
  Instructs GAMS/Sulum to use the priority branching information passed by GAMS through variable suffix values `variable.prior`.

## 5 Summary of SULUM Options

The GAMS/Sulum options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file `sulum.opt`.

```
simprimprice 2
optimizer 1
```

It will cause Sulum to use approximate steepest edge pricing strategy and will use the primal simplex algorithm.
### 5.1 Options related to the simplex optimizer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>kappa</td>
<td>Display approximate condition number estimates for the optimal simplex basis</td>
<td>0</td>
</tr>
<tr>
<td>mipmaxsimiter</td>
<td>Controls the maximum number of simplex iterations processed by the Sulum MIP optimizer.</td>
<td>maxint</td>
</tr>
<tr>
<td>simdualprice</td>
<td>Controls which pricing strategy should be used by the dual simplex optimizer.</td>
<td>0</td>
</tr>
<tr>
<td>simdualpricehotstart</td>
<td>Controls which pricing strategy should be used by the dual simplex optimizer, when a hotstart is available.</td>
<td>0</td>
</tr>
<tr>
<td>simloglevel</td>
<td>Controls the amounts of output from the simplex optimizer.</td>
<td>5</td>
</tr>
<tr>
<td>simmaxiter</td>
<td>Maximum iterations allowed in simplex optimizers.</td>
<td>infinity</td>
</tr>
<tr>
<td>simobjcutnosol</td>
<td>Controls if a solution is needed when the optimizer stops premature due to objective cut.</td>
<td>0</td>
</tr>
<tr>
<td>simobjlocut</td>
<td>If the optimal objective value can be proved to be less than this value the optimizer terminates.</td>
<td>mindouble</td>
</tr>
<tr>
<td>simobjupcut</td>
<td>If the optimal objective value can be proved to be larger than this value the optimizer terminates.</td>
<td>maxdouble</td>
</tr>
<tr>
<td>simperturblevel</td>
<td>Controls the level of perturbations in the simplex optimizer.</td>
<td>50</td>
</tr>
<tr>
<td>simprimprice</td>
<td>Controls which pricing strategy should be used by the primal simplex optimizer.</td>
<td>0</td>
</tr>
<tr>
<td>simprimpricehotstart</td>
<td>Controls which pricing strategy should be used by the primal simplex optimizer, when a hotstart is available.</td>
<td>0</td>
</tr>
<tr>
<td>simprob</td>
<td>Some time it might be faster to solve the respective dual formulation instead of the primal.</td>
<td>0</td>
</tr>
<tr>
<td>simquadprecision</td>
<td>Controls if quad precision is used in the simplex optimizer.</td>
<td>0</td>
</tr>
<tr>
<td>simscale</td>
<td>Controls if the simplex optimizer should scale data to be more numerical stable.</td>
<td>1</td>
</tr>
<tr>
<td>simscalehotstart</td>
<td>Controls if the simplex optimizer should scale data to be more numerical stable, when a hotstart is present.</td>
<td>1</td>
</tr>
<tr>
<td>simshifting</td>
<td>Controls if shifting is used in the simplex optimizer.</td>
<td>1</td>
</tr>
<tr>
<td>simsolveunscaled</td>
<td>Controls if the simplex optimizer should reoptimize on an unscaled problem if tolerances are not met.</td>
<td>1</td>
</tr>
<tr>
<td>simtimelimit</td>
<td>Maximum time allowed in the simplex optimizer.</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>simtolual</td>
<td>Absolute tolerance used by the simplex optimizer to determine if a solution is dual feasible or not.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>simtolmarko</td>
<td>Absolute tolerance used by the simplex optimizer to control the stability of pivot size in LU factorization module.</td>
<td>8.0e-3</td>
</tr>
<tr>
<td>simtolpivot</td>
<td>Absolute tolerance used by the simplex optimizer to control the minimum size of a pivot element.</td>
<td>1.0e-9</td>
</tr>
<tr>
<td>simtolprim</td>
<td>Absolute tolerance used by the simplex optimizer to determine if a solution is primal feasible or not.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>simusenetwork</td>
<td>Determines if the simplex optimizer will look for complete network structure and use the network optimizer when no hotstart is present.</td>
<td>1</td>
</tr>
<tr>
<td>simusenetworkhotstart</td>
<td>Determines if the simplex optimizer will look for complete network structure and use the network optimizer when a hotstart is present.</td>
<td>0</td>
</tr>
<tr>
<td>simusequadinf</td>
<td>Controls if the Sulum Simplex Optimizer should use switch to quad precision, when a problem is determined infeasible.</td>
<td>1</td>
</tr>
<tr>
<td>simwarmstart</td>
<td>If this key is switched off then the optimizer disregard any solution stored in the model.</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.2 Options related to the mixed integer optimizer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bndoptmip</td>
<td>Determines if the MIP formulations can be used in the bound optimizer module.</td>
<td>1</td>
</tr>
<tr>
<td>fixoptfile</td>
<td>Option file for fixed problem optimization</td>
<td></td>
</tr>
<tr>
<td>mipadvancedpseudocost</td>
<td>Controls if the Sulum MIP Optimizer should use advanced techniques in maintaining pseudo costs.</td>
<td>1</td>
</tr>
<tr>
<td>mipbigm</td>
<td>Maximum big M penalty used in MIP reformulations performed by the mixed integer optimizer.</td>
<td>1.0e+6</td>
</tr>
<tr>
<td>mipbranchselect</td>
<td>Controls how branching variables are selected in Sulum MIP optimizer.</td>
<td>0</td>
</tr>
<tr>
<td>mipbranchselectloglevel</td>
<td>Controls the amount of output from Sulum integer optimizer in the branch selection module.</td>
<td>0</td>
</tr>
<tr>
<td>mipcliquecuts</td>
<td>Controls the level of cut generation in clique cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>mipcoeffcuts</td>
<td>Controls the level of cut generation in tight coeff cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>mipcutoff</td>
<td>Cutoff value to be used to prune nodes in the mixed integer optimizer.</td>
<td>GAMS cutoff</td>
</tr>
<tr>
<td>mipcuts</td>
<td>Controls the overall level of cut generation used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>mipcutsinlppround</td>
<td>Controls the maximum number of cuts added in each reoptimizing round used by the Sulum MIP optimizer in cut reoptimization.</td>
<td>500</td>
</tr>
<tr>
<td>mipcutmaxadd</td>
<td>Maximum fraction of cuts added compared to number of constraints in the mixed integer optimizer.</td>
<td>0.5</td>
</tr>
<tr>
<td>mipcutoff</td>
<td>Cutoff value to be used to prune nodes in the mixed integer optimizer.</td>
<td>GAMS cutoff</td>
</tr>
<tr>
<td>mipdeterministic</td>
<td>Determines if the Sulum mixed Integer Optimizer should use deterministic mode.</td>
<td>1</td>
</tr>
<tr>
<td>mipdotiming</td>
<td>Controls if the mixed integer optimizer should do extra timing which can be fetched from varies information items.</td>
<td>0</td>
</tr>
<tr>
<td>mipfeasfocus</td>
<td>Controls if the mixed integer optimizer should focus more on obtaining a feasible solution.</td>
<td>0</td>
</tr>
<tr>
<td>mipflowcovercuts</td>
<td>Controls the level of cut generation in flow cover cuts used by the Sulum MIP optimizer (Not implemented yet).</td>
<td>4</td>
</tr>
<tr>
<td>mipgomorycuts</td>
<td>Controls the level of cut generation in Gomory cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>mipgomoryfractionalcuts</td>
<td>Controls the level of cut generation in Gomory fractional cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>miphotstart</td>
<td>Use previous MIP solutions when present.</td>
<td>1</td>
</tr>
<tr>
<td>miphotstartrins</td>
<td>Use RINS to repair previous solutions when a hotstart is present.</td>
<td>1</td>
</tr>
<tr>
<td>mipimpliebdoundcuts</td>
<td>Controls the level of cut generation in implied bound cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>mipintsolloglevel</td>
<td>Controls the amount of output from Sulum integer optimizer about finding integer feasible solutions.</td>
<td>0</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>mipknapsackcovercuts</td>
<td>Controls the level of cut generation in knapsack cover cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>miplocalnodecuts</td>
<td>Controls if the Sulum MIP Optimizer should use node cuts at local nodes i.e., nodes besides the root node.</td>
<td>10</td>
</tr>
<tr>
<td>miplocalnodeheuristic</td>
<td>Controls if the Sulum MIP Optimizer should use node heuristics at local nodes i.e., nodes besides the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodeheuristicdiving</td>
<td>Controls if the Sulum MIP Optimizer should use node diving heuristics at local nodes.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodeheuristicleargreedy</td>
<td>Controls if the Sulum MIP Optimizer should use node a greedy improvement heuristics at local nodes.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodeheuristiclocalsearch</td>
<td>Controls if the Sulum MIP Optimizer should use node local search improvement heuristics at local nodes.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodeheuristicpump</td>
<td>Controls if the Sulum MIP Optimizer should use node pumping heuristics at local nodes i.e., nodes besides the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodeheuristicrens</td>
<td>Controls if the Sulum MIP Optimizer should use node RENS(Relaxation Enforced Neighborhood Search) rounding heuristics at local nodes.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodeheuristicrins</td>
<td>Controls if the Sulum MIP Optimizer should use node RINS(Relaxation Induced Neighborhood Search) improvement heuristics at local nodes.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodeheuristicssins</td>
<td>Controls if the Sulum MIP Optimizer should use node SINS(Sulum Induced Neighborhood Search) improvement heuristics at local nodes.</td>
<td>1</td>
</tr>
<tr>
<td>miplocalnodepresolve</td>
<td>Controls if the Sulum MIP Optimizer should use node presolve at local nodes i.e., nodes besides the root node.</td>
<td>10</td>
</tr>
<tr>
<td>miplocalnodeprobing</td>
<td>Controls if the mixed integer optimizer can apply probing techniques in a local node.</td>
<td>1</td>
</tr>
<tr>
<td>miploglevel</td>
<td>Controls the amount of output from Sulum integer optimizer in general.</td>
<td>5</td>
</tr>
<tr>
<td>mipmaxnodes</td>
<td>Controls the maximum number of nodes processed by the Sulum MIP optimizer.</td>
<td>maxint</td>
</tr>
<tr>
<td>mipmaxrestarts</td>
<td>Controls the level of maximum number MIP restarts used by the Sulum MIP optimizer.</td>
<td>1</td>
</tr>
<tr>
<td>mipmaxsol</td>
<td>Controls the maximum number of feasible solutions found and stored in the solution pool by the Sulum MIP optimizer.</td>
<td>10</td>
</tr>
<tr>
<td>mipmaxvariability</td>
<td>Determines if the Sulum mixed Integer Optimizer should try to generate different paths, so the user can exploit variability.</td>
<td>0</td>
</tr>
<tr>
<td>mimpircuts</td>
<td>Controls the level of cut generation in MIR cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>mipnodechildselect</td>
<td>Controls how node selection on child nodes are selected in Sulum MIP optimizer.</td>
<td>0</td>
</tr>
<tr>
<td>mipnodeheuristiceasy</td>
<td>Controls if the Sulum MIP Optimizer should use easy heuristic in the tree nodes.</td>
<td>1</td>
</tr>
<tr>
<td>mipnodeheuristicswop</td>
<td>Controls if the Sulum MIP Optimizer should use swopping heuristic in the tree nodes.</td>
<td>1</td>
</tr>
<tr>
<td>mipnodeselect</td>
<td>Controls how nodes are selected in Sulum MIP optimizer.</td>
<td>0</td>
</tr>
<tr>
<td>mipnodeselectloglevel</td>
<td>Controls the amount of output from Sulum integer optimizer in the node selection module.</td>
<td>0</td>
</tr>
<tr>
<td>mipobjlocut</td>
<td>If the optimal objective value can be proved to be less than this value and a feasible solution exists the optimizer terminates.</td>
<td>mindouble</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>mipobjupcut</td>
<td>If the optimal objective value can be proved to be larger than this value and a feasible solution exists the optimizer terminates.</td>
<td>maxdouble</td>
</tr>
<tr>
<td>mipparallelrootsolve</td>
<td>Determines if the Sulum mixed Integer Optimizer should use parallel solves of multiple root nodes, exchange information between solves and then picking the best one.</td>
<td>1</td>
</tr>
<tr>
<td>mippresolverootprobing</td>
<td>Controls if the mixed integer optimizer should apply probing techniques in presolve.</td>
<td>1</td>
</tr>
<tr>
<td>mipreducedcoststrengthnode</td>
<td>Controls if the Sulum MIP Optimizer should use reduced cost strengthening at the each node.</td>
<td>1</td>
</tr>
<tr>
<td>mipreducedcoststrengthroot</td>
<td>Controls if the Sulum MIP Optimizer should use reduced cost strengthening at the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristicdiving</td>
<td>Controls if the Sulum MIP Optimizer should use node diving heuristics in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristiceasy</td>
<td>Controls if the Sulum MIP Optimizer should use easy heuristic in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristicgreedy</td>
<td>Controls if the Sulum MIP Optimizer should use node a greedy improvement heuristics in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristiclocaelsearch</td>
<td>Controls if the Sulum MIP Optimizer should use node local search improvement heuristics in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristicrens</td>
<td>Controls if the Sulum MIP Optimizer should use node RENS(Relaxation Enforced Neighborhood Search) rounding heuristics in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristicrcins</td>
<td>Controls if the Sulum MIP Optimizer should use node RENS(Relaxation Enforced Neighborhood Search) rounding heuristics in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristicstincs</td>
<td>Controls if the Sulum MIP Optimizer should use node SINS(Sulum Induced Neighborhood Search) improvement heuristics in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>miprootheuristicswop</td>
<td>Controls if the Sulum MIP Optimizer should use swopping heuristic in the root node.</td>
<td>1</td>
</tr>
<tr>
<td>mipsolstopnum</td>
<td>Controls if the mixed integer optimizer should stop after a certain number of integer solutions is found.</td>
<td>maxint</td>
</tr>
<tr>
<td>mipsolveaslp</td>
<td>Controls if the Sulum Optimizer should solve a MIP as LP i.e., relaxing integer constraints.</td>
<td>0</td>
</tr>
<tr>
<td>mipstalllimit</td>
<td>Controls if the mixed integer optimizer should be stopped prematurely due to no progress.</td>
<td>maxint</td>
</tr>
<tr>
<td>mipstronginit</td>
<td>Controls if the mixed integer optimizer should apply strong branching initialize in non strong branching rules.</td>
<td>1</td>
</tr>
<tr>
<td>mipsubmippinglevel</td>
<td>Controls how deep the mixed integer optimizer can go on solve sub mips.</td>
<td>1</td>
</tr>
<tr>
<td>miptimelimit</td>
<td>Maximum time allowed in the mixed integer optimizer.</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>miptolabsgap</td>
<td>Absolute stopping tolerance used by the MIP optimizer.</td>
<td>GAMS optca</td>
</tr>
<tr>
<td>miptolint</td>
<td>Integer variable tolerance used by the MIP optimizer.</td>
<td>1.0e-6</td>
</tr>
<tr>
<td>miptolerelgap</td>
<td>Relative stopping tolerance used by the MIP optimizer.</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td>miptwomircuts</td>
<td>Controls the level of cut generation in two MIR cuts used by the Sulum MIP optimizer.</td>
<td>4</td>
</tr>
<tr>
<td>mipusereformulation</td>
<td>Controls if the Sulum MIP optimizer is allowed to reformulate the MIP problem.</td>
<td>1</td>
</tr>
</tbody>
</table>
### 5.3 Options related to logging in Sulum

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>loglevel</td>
<td>Controls the amount of output from Sulum in general.</td>
<td>5</td>
</tr>
<tr>
<td>lognomodulemessage</td>
<td>Skip writing stop and start for each optimizer module in logging.</td>
<td>0</td>
</tr>
<tr>
<td>logprefix</td>
<td>Skip doing prefix in logging i.e., stream tags like info, debug, log, etc. is stripped.</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.4 Options related to the presolve module

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presolve</td>
<td>Controls which type of presolve strategy should be used by the presolve module.</td>
<td>1</td>
</tr>
<tr>
<td>presolvecompress</td>
<td>Controls if the problem should be compressed after a call to the presolve module.</td>
<td>0</td>
</tr>
<tr>
<td>presolvecompresshotstart</td>
<td>Controls if the problem should be compressed after a call to the presolve module were a hotstart is present.</td>
<td>0</td>
</tr>
<tr>
<td>presolvehotstart</td>
<td>Controls which type of presolve strategy should be used by the presolve module, when a hotstart is present.</td>
<td>1</td>
</tr>
</tbody>
</table>

### 5.5 Other options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bndoptlevel</td>
<td>Determines the level of effort used in the bound optimizer module.</td>
<td>1</td>
</tr>
<tr>
<td>bndoptmaxiter</td>
<td>Determines maximum of iterations which can be used in the bound optimizer module.</td>
<td>maxint</td>
</tr>
<tr>
<td>bndoptmaxnodes</td>
<td>Determines maximum of nodes which can be used in the bound optimizer module.</td>
<td>maxint</td>
</tr>
<tr>
<td>bndoptmaxnz</td>
<td>Determines maximum of non zeroes which can be used in the bound optimizer module.</td>
<td>maxint</td>
</tr>
<tr>
<td>debug</td>
<td>This option can be switched on in debug mode and development phase to find bugs easier.</td>
<td>0</td>
</tr>
<tr>
<td>dualcrash</td>
<td>Controls if the dual simplex optimizer should crash an advanced start basis.</td>
<td>1</td>
</tr>
<tr>
<td>dumpsolution</td>
<td>Controls export of alternate MIP solutions</td>
<td></td>
</tr>
<tr>
<td>lpreadfreecons</td>
<td>Controls if Sulum LP reader should read free constraints.</td>
<td>0</td>
</tr>
<tr>
<td>lpwritefreecons</td>
<td>Controls if Sulum LP writer should write free constraints.</td>
<td>0</td>
</tr>
<tr>
<td>lpwritenames</td>
<td>Controls if Sulum LP writer should replace constraint and variable names with generic ones.</td>
<td>0</td>
</tr>
<tr>
<td>lpwritenumonline</td>
<td>Controls how many items Sulum LP writer should write on each line.</td>
<td>5</td>
</tr>
<tr>
<td>lpwriteprecision</td>
<td>Controls how many items Sulum LP writer should write on each line.</td>
<td>4</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>lpwritevarorder</td>
<td>Controls if Sulum LP writer should write variables in same order, i.e., write variables with zero objective in objective section.</td>
<td>1</td>
</tr>
<tr>
<td>mpsreadfreecons</td>
<td>Controls if Sulum MPS reader should read free constraints.</td>
<td>0</td>
</tr>
<tr>
<td>mpswritefreecons</td>
<td>Controls if Sulum MPS writer should write free constraints.</td>
<td>0</td>
</tr>
<tr>
<td>mpswritenames</td>
<td>Controls if Sulum MPS writer should replace constraint and variable names with generic ones.</td>
<td>0</td>
</tr>
<tr>
<td>names</td>
<td>Indicator for loading names</td>
<td>1</td>
</tr>
<tr>
<td>numthreads</td>
<td>Controls the maximum number of threads used by the Sulum optimizer. If set to zero Sulum will use a number of threads corresponding to number of cores detected.</td>
<td>GAMS threads</td>
</tr>
<tr>
<td>optimizer</td>
<td>Controls which optimizer will be used.</td>
<td>0</td>
</tr>
<tr>
<td>optsolvezero</td>
<td>Tolerance on what is considered zero in solves with basis call by the user.</td>
<td>1.0e-12</td>
</tr>
<tr>
<td>optimelimit</td>
<td>Maximum time allowed in the optimizer.</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>primcrash</td>
<td>Controls if the primal simplex optimizer should crash an advanced start basis.</td>
<td>1</td>
</tr>
<tr>
<td>printoptions</td>
<td>List values of all options to GAMS listing file</td>
<td>1</td>
</tr>
<tr>
<td>randomseed</td>
<td>Controls the random seed by the Sulum MIP optimizer.</td>
<td>1234</td>
</tr>
<tr>
<td>rerun</td>
<td>Resolve without presolve in case of unbounded or infeasible</td>
<td>0</td>
</tr>
<tr>
<td>solwritenames</td>
<td>Controls if the Sulum solution writer should replace constraint and variable names with generic ones.</td>
<td>0</td>
</tr>
<tr>
<td>updatesolquality</td>
<td>Decides if the optimizer should update information solution quality items at the end of a call to the optimizer.</td>
<td>1</td>
</tr>
<tr>
<td>writenamemaxlength</td>
<td>Controls the maximum length of constraint and variable names the file writer will allow, before switching to generic names.</td>
<td>255</td>
</tr>
<tr>
<td>writeprob</td>
<td>Save the problem instance</td>
<td></td>
</tr>
<tr>
<td>wrnlargea</td>
<td>If an absolute value in the constraint matrix is larger than this value a warning will be displayed, but only if debug is on.</td>
<td>1.0e+8</td>
</tr>
<tr>
<td>wrnlargec</td>
<td>If an absolute value in the objective is larger than this value a warning will be displayed, but only if debug is on.</td>
<td>1.0e+8</td>
</tr>
<tr>
<td>wrnlargelo</td>
<td>If the absolute value of a lower bound is larger than this value a warning will be displayed, but only if debug is on.</td>
<td>1.0e+8</td>
</tr>
<tr>
<td>wrnlargeup</td>
<td>If the absolute value of a upper bound is larger than this value a warning will be displayed, but only if debug is on.</td>
<td>1.0e+8</td>
</tr>
<tr>
<td>wrnsmalla</td>
<td>If an absolute value in the constraint matrix is smaller than this value a warning will be displayed, but only if debug is on.</td>
<td>1.0e+8</td>
</tr>
</tbody>
</table>

### 6 GAMS/Sulum Log File

Sulum reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the simplex algorithms, each log line prints the iteration number, followed by infeasibility or objective value, and the elapsed wall clock time in every 5th line. The simplex screen log has the following appearance:

Starting optimizer
Checkout license of slmlp count 1 OK expires in : permanent
Starting presolve
Problem to be optimized has 2725 constraints 6569 variables and 36535 non zeroes in A matrix.
Presolved problem has 2079 constraints 5453 variables and 32948 non zeroes in A matrix.
Completed presolve
Presolve time : 0.119
Starting Simplex Optimizer
No hotstart used
Starting Dual Simplex Optimizer
Iters : 0 DInf : 4.9532861e+01 (289)
Increased marko tol from 8.0000000e-03 to 1.5000000e-02
Iters : 115 DInf : 4.1218938e+01 (53)
Iters : 226 DInf : 3.2040544e+01 (43)
Iters : 343 DObj : 1.1733784e+06 (1073)
Iters : 461 DObj : 5.7201242e+01 (1026) Time : 0.176
Iters : 583 DObj : 3.4110435e+05 (909)
Iters : 695 DObj : 2.5721816e+05 (835)
Iters : 812 DObj : 2.0755088e+05 (739)
Iters : 930 DObj : 1.7895719e+05 (640)
Iters : 1054 DObj : 1.5940993e+05 (637) Time : 0.231
Iters : 1170 DObj : 1.4558622e+05 (592)
Iters : 1287 DObj : 1.4006841e+05 (552)
Iters : 1420 DObj : 1.3044226e+05 (469)
Iters : 1538 DObj : 1.2800187e+05 (417)
Iters : 1677 DObj : 1.2092364e+05 (454) Time : 0.300
Iters : 1794 DObj : 1.1953061e+05 (389)
Iters : 1904 DObj : 1.1932819e+05 (326)
Iters : 2025 DObj : 1.1829430e+05 (210)
Iters : 2135 DObj : 1.1792865e+05 (216)
Iters : 2245 DObj : 1.1775868e+05 (187) Time : 0.388
Iters : 2356 DObj : 1.1758897e+05 (237)
Iters : 2466 DObj : 1.1758618e+05 (240)
Iters : 2576 DObj : 1.1756110e+05 (158)
Iters : 2686 DObj : 1.1742065e+05 (218)
Iters : 2796 DObj : 1.1702324e+05 (246) Time : 0.495
Iters : 2907 DObj : 1.1668466e+05 (246)
Iters : 3017 DObj : 1.1632811e+05 (168)
Iters : 3127 DObj : 1.1604803e+05 (186)
Iters : 3237 DObj : 1.1579318e+05 (112)
Iters : 3347 DObj : 1.1536583e+05 (92) Time : 0.710
Iters : 3457 DObj : 1.1505896e+05 (67)
Iters : 3567 DObj : 1.1487703e+05 (16)
Solution is optimal
Completed Dual Simplex Optimizer
Completed Simplex Optimizer
Starting postsolve
Completed postsolve
Solution needs cleaning
Improving numerical precision
Iters : 3584 DObj : 1.1487366e+05 (0)
Solution is optimal
Completed Dual Simplex Optimizer
Completed Simplex Optimizer
Completed optimizer
Optimizer Time : 0.837
Setup solution quality items
Model was solved to optimality (subject to tolerances).
7 Detailed Descriptions of SULUM Options

\textbf{bndoptlevel (integer):} Determines the level of effort used in the bound optimizer module.

\hspace{1em} (default = 1)
\hspace{2em} 1 Use moderate effort in bound optimizer module.
\hspace{2em} 2 Use aggressive effort in bound optimizer module.
\hspace{2em} 3 Use maximum effort in bound optimizer module.

\textbf{bndoptmaxiter (integer):} Determines maximum of iterations which can be used in the bound optimizer module.

\hspace{1em} (default = maxint)

\textbf{bndoptmaxnodes (integer):} Determines maximum of nodes which can be used in the bound optimizer module.

\hspace{1em} (default = maxint)

\textbf{bndoptmaxnz (integer):} Determines maximum of non zeroes which can be used in the bound optimizer module.

\hspace{1em} (default = maxint)

\textbf{bndoptmip (integer):} Determines if the MIP formulations can be used in the bound optimizer module.

\hspace{1em} (default = 1)
\hspace{2em} 0 The given option is off.
\hspace{2em} 1 The given option is on.

\textbf{debug (integer):} This option can be switched on in debug mode and development phase to find bugs easier.

\hspace{1em} (default = 0)
\hspace{2em} 0 The given option is off.
\hspace{2em} 1 The given option is on.

\textbf{dualcrash (integer):} Controls if the dual simplex optimizer should crash an advanced start basis.

\hspace{1em} (default = 1)
\hspace{2em} 0 The given option is off.
\hspace{2em} 1 The given option is on.

\textbf{dumpsolution (string):} Controls export of alternate MIP solutions

\textbf{fixoptfile (string):} Option file for fixed problem optimization

\textbf{kappa (integer):} Display approximate condition number estimates for the optimal simplex basis

\hspace{1em} (default = 0)
\hspace{2em} 0 Do not compute and display approximate condition number
\hspace{2em} 1 Compute and display approximate condition number

\textbf{loglevel (integer):} Controls the amount of output from Sulum in general.

\hspace{1em} (default = 5)

\textbf{lognomodulemessage (integer):} Skip writing stop and start for each optimizer module in logging.

\hspace{1em} (default = 0)
\hspace{2em} 0 The given option is off.
\hspace{2em} 1 The given option is on.

\textbf{logprefix (integer):} Skip doing prefix in logging i.e., stream tags like info, debug, log, etc. is stripped.

\hspace{1em} (default = 0)
\hspace{2em} 0 The given option is off.
1. The given option is on.

**lpreadfreecons** *(integer)*: Controls if Sulum LP reader should read free constraints.

(default = 0)

0 The given option is off.
1 The given option is on.

**lpwritefreecons** *(integer)*: Controls if Sulum LP writer should write free constraints.

(default = 0)

0 The given option is off.
1 The given option is on.

**lpwritenames** *(integer)*: Controls if Sulum LP writer should replace constraint and variable names with generic ones.

(default = 0)

0 The given option is off.
1 The given option is on.

**lpwritenumonline** *(integer)*: Controls how many items Sulum LP writer should write on each line.

(default = 5)

**lpwriteprecision** *(integer)*: Controls how many items Sulum LP writer should write on each line.

(default = 4)

**lpwritevarorder** *(integer)*: Controls if Sulum LP writer should write variables in same order, i.e., write variables with zero objective in objective section.

(default = 1)

0 The given option is off.
1 The given option is on.

**mipadvancedpseudocost** *(integer)*: Controls if the Sulum MIP Optimizer should use advanced techniques in maintaining pseudo costs.

(default = 1)

0 The given option is off.
1 The given option is on.

**mipbignm** *(real)*: Maximum big M penalty used in MIP reformulations performed by the mixed integer optimizer.

(default = 1.0e+6)

**mipbranchselect** *(integer)*: Controls how branching variables are selected in Sulum MIP optimizer.

(default = 0)

0 The given option is automatically determined by the optimizer.
1 Choose based on minimum infeasibility.
2 Choose based on maximum infeasibility.
3 Choose based on strong branching.
4 Choose based on pseudo costs.
5 Choose based on a hybrid pseudo price selection scheme.

**mipbranchselectlogfile** *(integer)*: Controls the amount of output from Sulum integer optimizer in the branch selection module.

(default = 0)

**mipcliquecuts** *(integer)*: Controls the level of cut generation in clique cuts used by the Sulum MIP optimizer.
mipcoeffcuts (integer): Controls the level of cut generation in tight coeff cuts used by the Sulum MIP optimizer.

(default = 4)

0 The given option is off i.e., do no cut generation.
1 Do a minimum of work in cut generation.
2 Do a fair amount of work in cut generation.
3 Be aggressive in cut generation.
4 The given option is automatically determined by the optimizer.

mipcutmaxadd (real): Maximum fraction of cuts added compared to number of constraints in the mixed integer optimizer.

(default = 0.5)

mipcutoff (real): Cutoff value to be used to prune nodes in the mixed integer optimizer.

Range: [mindouble, maxdouble]

(default = GAMS cutoff)

mipcuts (integer): Controls the overall level of cut generation used by the Sulum MIP optimizer.

(default = 4)

0 The given option is off i.e., do no cut generation.
1 Do a minimum of work in cut generation.
2 Do a fair amount of work in cut generation.
3 Be aggressive in cut generation.
4 The given option is automatically determined by the optimizer.

mipcutsinlpprround (integer): Controls the maximum number of cuts added in each reoptimizing round used by the Sulum MIP optimizer in cut reoptimization.

(default = 500)

mipcutsuselexdual (integer): Controls if lexicographic dual simplex is used by the Sulum MIP optimizer in cut reoptimization.

(default = 1)

0 The given option is off.
1 The given option is on.

mipdeterministic (integer): Determines if the Sulum mixed Integer Optimizer should use deterministic mode.

(default = 1)

0 The given option is off.
1 The given option is on.

mipdotiming (integer): Controls if the mixed integer optimizer should do extra timing which can be fetched from varies information items.

(default = 0)

0 The given option is off.
1. The given option is on.

**mipfeasfocus (integer):** Controls if the mixed integer optimizer should focus more on obtaining a feasible solution.

(default = 0)

0. The given option is off.
1. The given option is on.

**mipflowcovercuts (integer):** Controls the level of cut generation in flow cover cuts used by the Sulum MIP optimizer (Not implemented yet).

(default = 4)

0. The given option is off i.e., do no cut generation.
1. Do a minimum of work in cut generation.
2. Do a fair amount of work in cut generation.
3. Be aggressive in cut generation.
4. The given option is automatically determined by the optimizer.

**mipgomorycuts (integer):** Controls the level of cut generation in Gomory cuts used by the Sulum MIP optimizer.

(default = 4)

0. The given option is off i.e., do no cut generation.
1. Do a minimum of work in cut generation.
2. Do a fair amount of work in cut generation.
3. Be aggressive in cut generation.
4. The given option is automatically determined by the optimizer.

**mipgomoryfractionalcuts (integer):** Controls the level of cut generation in Gomory fractional cuts used by the Sulum MIP optimizer.

(default = 4)

0. The given option is off i.e., do no cut generation.
1. Do a minimum of work in cut generation.
2. Do a fair amount of work in cut generation.
3. Be aggressive in cut generation.
4. The given option is automatically determined by the optimizer.

**miphotstart (integer):** Use previous MIP solutions when present.

(default = 1)

0. The given option is off.
1. The given option is on.

**miphotstartrins (integer):** Use RINS to repair previous solutions when a hotstart is present.

(default = 1)

0. The given option is off.
1. The given option is on.

**mipimpliedboundcuts (integer):** Controls the level of cut generation in implied bound cuts used by the Sulum MIP optimizer.

(default = 4)

0. The given option is off i.e., do no cut generation.
1. Do a minimum of work in cut generation.
2 Do a fair amount of work in cut generation.
3 Be aggressive in cut generation.
4 The given option is automatically determined by the optimizer.

**mipintsolloglevel (integer):** Controls the amount of output from Sulum integer optimizer about finding integer feasible solutions.

(default = 0)

**mipknapsackcovercuts (integer):** Controls the level of cut generation in knapsack cover cuts used by the Sulum MIP optimizer.

(default = 4)
0 The given option is off i.e., do no cut generation.
1 Do a minimum of work in cut generation.
2 Do a fair amount of work in cut generation.
3 Be aggressive in cut generation.
4 The given option is automatically determined by the optimizer.

**miplocalnodecuts (integer):** Controls if the Sulum MIP Optimizer should use node cuts at local nodes i.e., nodes besides the root node.

(default = 10)

**miplocalnodeheuristic (integer):** Controls if the Sulum MIP Optimizer should use node heuristics at local nodes i.e., nodes besides the root node.

(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miplocalnodeheuristicdiving (integer):** Controls if the Sulum MIP Optimizer should use node diving heuristics at local nodes.

(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miplocalnodeheuristiclocalgreedy (integer):** Controls if the Sulum MIP Optimizer should use node a greedy improvement heuristics at local nodes.

(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miplocalnodeheuristiclocalsearch (integer):** Controls if the Sulum MIP Optimizer should use node local search improvement heuristics at local nodes.

(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

\textbf{miplocalnodeheuristicpump} (integer): Controls if the Sulum MIP Optimizer should use node pumping heuristics at local nodes i.e., nodes besides the root node.

\begin{verbatim}
(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.
\end{verbatim}

\textbf{miplocalnodeheuristicrens} (integer): Controls if the Sulum MIP Optimizer should use node RENS(Relaxation Enforced Neighborhood Search) rounding heuristics at local nodes.

\begin{verbatim}
(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.
\end{verbatim}

\textbf{miplocalnodeheuristicrins} (integer): Controls if the Sulum MIP Optimizer should use node RINS(Relaxation Induced Neighborhood Search) improvement heuristics at local nodes.

\begin{verbatim}
(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.
\end{verbatim}

\textbf{miplocalnodeheuristicsins} (integer): Controls if the Sulum MIP Optimizer should use node SINS(Sulum Induced Neighborhood Search) improvement heuristics at local nodes.

\begin{verbatim}
(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.
\end{verbatim}

\textbf{miplocalnodepresolve} (integer): Controls if the Sulum MIP Optimizer should use node presolve at local nodes i.e., nodes besides the root node.

\begin{verbatim}
(default = 10)
\end{verbatim}

\textbf{miplocalnodeprobing} (integer): Controls if the mixed integer optimizer can apply probing techniques in a local node.

\begin{verbatim}
(default = 1)
0 The given option is off.
1 The given option is on.
\end{verbatim}

\textbf{miploglevel} (integer): Controls the amount of output from Sulum integer optimizer in general.

\begin{verbatim}
(default = 5)
\end{verbatim}

\textbf{mipmaxnodes} (integer): Controls the maximum number of nodes processed by the Sulum MIP optimizer.
mipmaxrestarts (integer): Controls the level of maximum number MIP restarts used by the Sulum MIP optimizer.
(default = 1)

mipmaxsimiter (integer): Controls the maximum number of simlex iterations processed by the Sulum MIP optimizer.
(default = maxint)

mipmaxsol (integer): Controls the maximum number of feasible solutions found and stored in the solution pool by the Sulum MIP optimizer.
(default = 10)

mipmaxvariability (integer): Determines if the Sulum mixed Integer Optimizer should try to generate different paths, so the user can exploit variability.
(default = 0)
  0 The given option is off.
  1 The given option is on.

mipmircuts (integer): Controls the level of cut generation in MIR cuts used by the Sulum MIP optimizer.
(default = 4)
  0 The given option is off i.e., do no cut generation.
  1 Do a minimum of work in cut generation.
  2 Do a fair amount of work in cut generation.
  3 Be aggressive in cut generation.
  4 The given option is automatically determined by the optimizer.

mipnodechildselect (integer): Controls how node selection on child nodes are selected in Sulum MIP optimizer.
(default = 0)
  0 The given option is automatically determined by the optimizer.
  1 Choose up branch.
  2 Choose lo branch.
  3 Choose branch guided by incumbent.

mipnodeheuristicseasy (integer): Controls if the Sulum MIP Optimizer should use easy heuristic in the tree nodes.
(default = 1)
  0 Do not apply heuristic method.
  1 The optimizer automatically decides if the heuristic method should be applied.
  2 Heuristic will be used in a lightweight version.
  3 Heuristic will be used as much as deemed possible.

mipnodeheuristicswop (integer): Controls if the Sulum MIP Optimizer should use swopping heuristic in the tree nodes.
(default = 1)
  0 Do not apply heuristic method.
  1 The optimizer automatically decides if the heuristic method should be applied.
  2 Heuristic will be used in a lightweight version.
  3 Heuristic will be used as much as deemed possible.

mipnodeselect (integer): Controls how nodes are selected in Sulum MIP optimizer.
(default = 0)
0 The given option is automatically determined by the optimizer.
1 Choose best bound first.
2 Choose lowest depth first.
3 Choose by pseudo costs.
4 Choose from a hybrid estimate.
5 Choose from an adaptive node search aim to balance the tree exploration.

**mipnodeselectloglevel** *(integer)*: Controls the amount of output from Sulum integer optimizer in the node selection module.

(default = 0)

**mipobjlocut** *(real)*: If the optimal objective value can be proved to be less than this value and a feasible solution exists the optimizer terminates.

(default = mindouble)

**mipobjupcut** *(real)*: If the optimal objective value can be proved to be larger than this value and a feasible solution exists the optimizer terminates.

(default = maxdouble)

**mipparallelrootsolve** *(integer)*: Determines if the Sulum mixed Integer Optimizer should use parallel solves of multiple root nodes, exchange information between solves and then picking the best one.

(default = 1)

0 The given option is off.
1 The given option is on.

**mippresolverootprobing** *(integer)*: Controls if the mixed integer optimizer should apply probing techniques in presolve.

(default = 1)

0 The given option is off.
1 The given option is on.

**mipreducedcoststrengthnode** *(integer)*: Controls if the Sulum MIP Optimizer should use reduced cost strengthening at the each node.

(default = 1)

0 The given option is off.
1 The given option is on.

**mipreducedcoststrengthroot** *(integer)*: Controls if the Sulum MIP Optimizer should use reduced cost strengthening at the root node.

(default = 1)

0 The given option is off.
1 The given option is on.

**miprootheuristicdiving** *(integer)*: Controls if the Sulum MIP Optimizer should use node diving heuristics in the root node.

(default = 1)

0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootheuristiceasy** *(integer)*: Controls if the Sulum MIP Optimizer should use easy heuristic in the root node.

(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootheuristicgreedy (integer):** Controls if the Sulum MIP Optimizer should use node a greedy improvement heuristics in the root node.

(default = 1)

0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootheuristiclocalsearch (integer):** Controls if the Sulum MIP Optimizer should use node local search improvement heuristics in the root node.

(default = 1)

0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootheuristicrens (integer):** Controls if the Sulum MIP Optimizer should use node RENS(Relaxation Enforced Neighborhood Search) rounding heuristics in the root node.

(default = 1)

0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootheuristicrins (integer):** Controls if the Sulum MIP Optimizer should use node RENS(Relaxation Enforced Neighborhood Search) rounding heuristics in the root node.

(default = 1)

0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootheuristicsins (integer):** Controls if the Sulum MIP Optimizer should use node SINS(Sulum Induced Neighborhood Search) improvement heuristics in the root node.

(default = 1)

0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootheuristicswop (integer):** Controls if the Sulum MIP Optimizer should use swopping heuristic in the root node.

(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**miprootnodeheuristicpump (integer):** Controls if the Sulum MIP Optimizer should use node pumping heuristics at the root node.

(default = 1)
0 Do not apply heuristic method.
1 The optimizer automatically decides if the heuristic method should be applied.
2 Heuristic will be used in a lightweight version.
3 Heuristic will be used as much as deemed possible.

**mipsolstopnum (integer):** Controls if the mixed integer optimizer should stop after a certain number of integer solutions is found.

(default = maxint)

**mipsolveaslp (integer):** Controls if the Sulum Optimizer should solve a MIP as LP i.e., relaxing integer constraints.

(default = 0)
0 The given option is off.
1 The given option is on.

**mipstalllimit (integer):** Controls if the mixed integer optimizer should be stopped prematurely due to no progress.

(default = maxint)

**mipstronginit (integer):** Controls if the mixed integer optimizer should apply strong branching initialize in non strong branching rules.

(default = 1)
0 The given option is off.
1 The given option is on.

**mipsubmippinglevel (integer):** Controls how deep the mixed integer optimizer can go on solve sub mips.

(default = 1)

**miptimelimit (real):** Maximum time allowed in the mixed integer optimizer.

(default = GAMS reslim)

**miptolabsgap (real):** Absolute stopping tolerance used by the MIP optimizer.

Range: [0.0, maxdouble]

(default = GAMS optca)

**miptolint (real):** Integer variable tolerance used by the MIP optimizer.

Range: [1.0e-9, 1.0e-2]

(default = 1.0e-6)

**miptolrelgap (real):** Relative stopping tolerance used by the MIP optimizer.

Range: [1.0e-16, maxdouble]

(default = GAMS optcr)

**miptwomircuts (integer):** Controls the level of cut generation in two MIR cuts used by the Sulum MIP optimizer.

(default = 4)
0 The given option is off i.e., do no cut generation.
1 Do a minimum of work in cut generation.
2 Do a fair amount of work in cut generation.
3 Be aggressive in cut generation.
4 The given option is automatically determined by the optimizer.

**mipuserereformulation** *(integer)*: Controls if the Sulum MIP optimizer is allowed to reformulate the MIP problem.

(default = 1)
0 The given option is off.
1 The given option is on.

**mipzerohalfcuts** *(integer)*: Controls the level of cut generation in zero half cuts used by the Sulum MIP optimizer (Not implemented yet).

(default = 4)
0 The given option is off i.e., do no cut generation.
1 Do a minimum of work in cut generation.
2 Do a fair amount of work in cut generation.
3 Be aggressive in cut generation.
4 The given option is automatically determined by the optimizer.

**mpsreadfreecons** *(integer)*: Controls if Sulum MPS reader should read free constraints.

(default = 0)
0 The given option is off.
1 The given option is on.

**mpswritefreecons** *(integer)*: Controls if Sulum MPS writer should write free constraints.

(default = 0)
0 The given option is off.
1 The given option is on.

**mpswritenames** *(integer)*: Controls if Sulum MPS writer should replace constraint and variable names with generic ones.

(default = 0)
0 The given option is off.
1 The given option is on.

**names** *(integer)*: Indicator for loading names

(default = 1)

**numthreads** *(integer)*: Controls the maximum number of threads used by the Sulum optimizer. If set to zero Sulum will use a number of threads corresponding to number of cores detected.

(default = GAMS threads)

**optimizer** *(integer)*: Controls which optimizer will be used.

(default = 0)
0 The optimizer decides which optimizer to call based on the model structure.
1 The primal simplex optimizer should be applied.
2 The dual simplex optimizer should be applied.

**optsolvezero** *(real)*: Tolerance on what is considered zero in solves with basis call by the user.

Range: [0, 1×10^-8]
**optimelimit** *(real):* Maximum time allowed in the optimizer.
(default = GAMS reslim)

**Presolve** *(integer):* Controls which type of presolve strategy should be used by the presolve module.
(default = 1)
0 Do not apply any presolve strategies.
1 The optimizer automatically decides if presolve and which type of presolve should be applied.
2 Presolve will only use very simple methods to reduce problem size.
3 The presolve will only use strategies based on primal information.
4 The presolve will only use strategies based on dual information.
5 Presolve will reduce problem size using all methods.

**presolvecompress** *(integer):* Controls if the problem should be compressed after a call to the presolve module.
(default = 0)
0 The presolve automatically decides if final problem should be compressed.
1 Presolve will use compression.
2 Do not apply any presolve compression.

**presolvecompresshotstart** *(integer):* Controls if the problem should be compressed after a call to the presolve module were a hotstart is present.
(default = 0)
0 The presolve automatically decides if final problem should be compressed.
1 Presolve will use compression.
2 Do not apply any presolve compression.

**presolvehotstart** *(integer):* Controls which type of presolve strategy should be used by the presolve module, when a hotstart is present.
(default = 1)
0 Do not apply any presolve strategies.
1 The optimizer automatically decides if presolve and which type of presolve should be applied.
2 Presolve will only use very simple methods to reduce problem size.
3 The presolve will only use strategies based on primal information.
4 The presolve will only use strategies based on dual information.
5 Presolve will reduce problem size using all methods.

**primcrash** *(integer):* Controls if the primal simplex optimizer should crash an advanced start basis.
(default = 1)
0 The given option is off.
1 The given option is on.

**printoptions** *(integer):* List values of all options to GAMS listing file
(default = 0)

**randomseed** *(integer):* Controls the random seed by the Sulum MIP optimizer.
(default = 1234)

**rerun** *(integer):* Resolve without presolve in case of unbounded or infeasible
SULUM 2.0

(simdualprice (integer): Controls which pricing strategy should be used by the dual simplex optimizer.
(default = 0)

0 The simplex optimizer analyzes the model and decides the best choice in the given situation.
1 The simplex optimizer will use steepest edge strategy, which is the most expensive pricing strategy, but also often the one with fewest iterations.
2 The simplex optimizer will use approximate steepest edge strategy, which relaxes the steepest edge strategy by using only approximate norms.
3 The simplex optimizer will scan only a subset of candidates and choose between promising candidates by a very cheap scheme.

(simdualpricehotstart (integer): Controls which pricing strategy should be used by the dual simplex optimizer, when a hotstart is available.
(default = 0)

0 The simplex optimizer analyzes the model and decides the best choice in the given situation.
1 The simplex optimizer will use steepest edge strategy, which is the most expensive pricing strategy, but also often the one with fewest iterations.
2 The simplex optimizer will use approximate steepest edge strategy, which relaxes the steepest edge strategy by using only approximate norms.
3 The simplex optimizer will scan only a subset of candidates and choose between promising candidates by a very cheap scheme.

(simloglevel (integer): Controls the amounts of output from the simplex optimizer.
(default = 5)

(simmaxiter (integer): Maximum iterations allowed in simplex optimizers.
(default = infinity)

(simobjcutnosol (integer): Controls if a solution is needed when the optimizer stops premature due to objective cut.
(default = 0)

0 The given option is off.
1 The given option is on.

(simobjlocut (real): If the optimal objective value can be proved to be less than this value the optimizer terminates.
(default = mindouble)

(simobjupcut (real): If the optimal objective value can be proved to be larger than this value the optimizer terminates.
(default = maxdouble)

(simperturblevel (integer): Controls the level of perturbations in the simplex optimizer.
(default = 50)

(simprimprice (integer): Controls which pricing strategy should be used by the primal simplex optimizer.
(default = 0)

0 The simplex optimizer analyzes the model and decides the best choice in the given situation.
1 The simplex optimizer will use steepest edge strategy, which is the most expensive pricing strategy, but also often the one with fewest iterations.
2 The simplex optimizer will use approximate steepest edge strategy, which relaxes the steepest edge strategy by using only approximate norms.
3 The simplex optimizer will scan only a subset of candidates and choose between promising candidates by a very cheap scheme.
simprimpricehotstart (integer): Controls which pricing strategy should be used by the primal simplex optimizer, when a hotstart is available.

(default = 0)

0 The simplex optimizer analyzes the model and decides the best choice in the given situation.
1 The simplex optimizer will use steepest edge strategy, which is the most expensive pricing strategy, but also often the one with fewest iterations.
2 The simplex optimizer will use approximate steepest edge strategy, which relaxes the steepest edge strategy by using only approximate norms.
3 The simplex optimizer will scan only a subset of candidates and choose between promising candidates by a very cheap scheme.

simprob (integer): Some time it might be faster to solve the respective dual formulation instead of the primal.

(default = 0)

0 The optimizer decides if the primal or dual formulation should be solved.
1 The primal formulation should be solved.
2 The dual formulation should be solved.

simquadprecision (integer): Controls if quad precision is used in the simplex optimizer.

(default = 0)

0 The given option is off.
1 The given option is on.

simscale (integer): Controls if the simplex optimizer should scale data to be more numerical stable.

(default = 1)

0 The given option is off.
1 The given option is on.

simscalehotstart (integer): Controls if the simplex optimizer should scale data to be more numerical stable, when a hotstart is present.

(default = 1)

0 The given option is off.
1 The given option is on.

simshifting (integer): Controls if shifting is used in the simplex optimizer.

(default = 1)

0 The given option is off.
1 The given option is on.

simsolveunscaled (integer): Controls if the simplex optimizer should reoptimize on a unscaled problem if tolerances are not met.

(default = 1)

0 The given option is off.
1 The given option is on.

simtimelimit (real): Maximum time allowed in the simplex optimizer.

(default = GAMS reslim)

sintoldual (real): Absolute tolerance used by the simplex optimizer to determine if a solution is dual feasible or not.

Range: [1.0e-10, 1.0e-4]
(default = 1.0e-6)

**simtolmarko (real):** Absolute tolerance used by the simplex optimizer to control the stability of pivot size in LU factorization module.

Range: [1.0e-4, 9.0e-1]
(default = 8.0e-3)

**simtolpivot (real):** Absolute tolerance used by the simplex optimizer to control the minimum size of a pivot element.

Range: [1.0e-12, 1.0e-5]
(default = 1.0e-9)

**simtolprim (real):** Absolute tolerance used by the simplex optimizer to determine if a solution is primal feasible or not.

Range: [1.0e-10, 1.0e-4]
(default = 1.0e-6)

**simusenetwork (integer):** Determines if the simplex optimizer will look for complete network structure and use the network optimizer when no hotstart is present.

(default = 1)
0 The given option is off.
1 The given option is on.

**simusenetworkhotstart (integer):** Determines if the simplex optimizer will look for complete network structure and use the network optimizer when a hotstart is present.

(default = 0)
0 The given option is off.
1 The given option is on.

**simusequadinf (integer):** Controls if the Sulum Simplex Optimizer should use switch to quad precision, when a problem is determined infeasible.

(default = 1)
0 The given option is off.
1 The given option is on.

**simwarmstart (integer):** If this key is switched off then the optimizer disregard any solution stored in the model.

(default = 1)
0 The given option is off.
1 The given option is on.

**solvefixed (integer):** Indicator for solving the fixed problem for a MIP to get a dual solution

(default = 1)

**solvtrace (string):** Filename of solving trace file

**solvetracenode (integer):** Node interval when a trace record is written

(default = 100)

**solvetracetimereal: Time interval when a trace record is written

(default = 1)

**solwritenames (integer):** Controls if the Sulum solution writer should replace constraint and variable names with generic ones.

(default = 0)
0 The given option is off.
1 The given option is on.

**updatesolquality** *(integer)*: Decides if the optimizer should update information solution quality items at the end of a call to the optimizer.

(default = 1)

0 The given option is off.

1 The given option is on.

**usebasis** *(integer)*: Use basis from GAMS

(default = GAMS bratio)

**writenamemaxlength** *(integer)*: Controls the maximum length of constraint and variable names the file writer will allow, before switching to generic names.

(default = 255)

**writeprob** *(string)*: Save the problem instance

**wrnlargea** *(real)*: If an absolute value in the constraint matrix is larger than this value a warning will be displayed, but only if debug is on.

(default = 1.0e+8)

**wrnlargec** *(real)*: If an absolute value in the objective is larger than this value a warning will be displayed, but only if debug is on.

(default = 1.0e+8)

**wrnlargelo** *(real)*: If the absolute value of a lower bound is larger than this value a warning will be displayed, but only if debug is on.

(default = 1.0e+8)

**wrnlargeup** *(real)*: If the absolute value of a upper bound is larger than this value a warning will be displayed, but only if debug is on.

(default = 1.0e+8)

**wrnsmalla** *(real)*: If an absolute value in the constraint matrix is smaller than this value a warning will be displayed, but only if debug is on.

(default = 1.0e+8)
1 Introduction

This document describes the GAMS/XA linear and mixed-integer programming solver. The GAMS/XA solver (here also simply referred to as XA) is based on Sunset Software Technology’s XA Callable Library, an implementation of high performance solvers for LP and MIP problems.

XA implements primal simplex, dual simplex, and barrier algorithms for solving linear problems. The primal/dual simplex method is very robust, and in most cases you should get good performance, especially from a warm start. The barrier method is particularly efficient on large models. Both algorithms benefit from XA’s presolver, which reduces the size of the model by removing redundant contraints, substituting constraints, etc.

In most cases, GAMS/XA should perform satisfactorily without using any options. However, if fine-tuning is necessary or desired, XA provides many options and parameters designed for this purpose. These options are accessible via GAMS option statements or via an XA-specific option file.

2 Usage

If you have installed the GAMS system and configured XA as the default LP, RMIP and MIP solver, all LP, RMIP and MIP models without a specific solver option will use XA. If you installed another solver as the default, you can explicitly request that a particular model be solved by XA by inserting the statement

```plaintext
option LP = xa; { or MIP or RMIP }
```

somewhere before the `solve` statement.
3 Memory Usage

By default, the GAMS/XA link computes an estimate of the amount of memory that will be required by the solver, and passes this on to the solver. The solver makes an allocation of this amount and then uses this memory during the course of program execution. Usually, this will be sufficient to solve the problem successfully. In some cases, though, the computed estimate will be too small, and GAMS/XA will indicate that a larger memory estimate is required. You will need to manually specify a larger memory estimate to solve the model.

A model-specified memory estimate can be made by adding the following line to your GAMS model before the solve statement:

```gams
<modelname>.workspace = xx;
```

where xx is the amount of memory in Mbytes. You can also define the environment variable XAMEMORY to be the amount of memory to use, in Mbytes. The computed memory estimate is the default, and is used only if no manual estimate is specified. The model-specified workspace limit overrides the computed estimate, and the XAMEMORY environment variable takes precedence over both of these.

In an attempt to insure that all models solve without running out of memory, XA makes one final memory check and if the user supplied memory amount is below what XA would consider reasonable for that size of problem, XA will then increase your amount to XA's minimal value.

On multi-processor machines, XA will automatically detect and use all available processors (CPU's) when solving MIP models. The memory estimate computed adds 50% more memory per processor to take full advantage of these processors, but this is sometimes not enough memory for XA to multi-process. In this case, a larger estimate must be specified manually.

4 Semi-Continuous and Semi-Integer Variables

XA supports semi-continuous and semi-integer variable types. Semi-continuous variables are variables that are either at zero or greater than or equal to their lower bound. E.g. a pump motor if operating must run between 2400 and 5400 r.p.m., but it may be switched off as well. Investment levels must exceed a specific threshold or no investment is made.

All semi-continuous variables must have a lower bound specification, e.g., `speed.lo(i) = 100`. Semi-integer variables must have an upper bound as well.

Prior to the introduction of these variable types, semi-continuous variables had to be emulated by adding one additional binary variable and one additional constraint for each semi-continuous variable. For models of any size, this approach very quickly increased the model's size beyond solvability. Now XA has implicitly defined these variables without requiring the addition of new variables and constraints to your model. This effectively increases the size of model that can be solved and does it in a very neat and clean way besides.

For example, to define variables 'a' and 'b' as semi-continuous enter:

```gams
SemiCont   a , b ;
```

or to define semi-integer variables -

```gams
SemiInt    y1 , y2 ;
```

Priority values (\(prior\) suffix) can be associated with both semi-continuous and semi-integer variables. All the integer solving options are available for models with semi-continuous and semi-integer variables as well. For example, you can select solving strategies, `optcr` and `optca` values, etc.

The solve time complexity for semi-continuous variables is comparable with the solve times for binary models, while the semi-integer case compares to integer.
5 Branch and Bound Topics

XA is designed to solve a vast majority of LP problems using the default settings. In the integer case, however, the default settings may not result in optimal speed and reliability. By experimenting with the control parameters performance can be improved (or worsened!) dramatically.

5.1 Branching Priorities

Using priorities can significantly reduce the amount of time required to obtain a good integer solution. If your model has a natural order in time, or in space, or in any other dimension then you should consider using priority branching. For example, multi-period production problem with inventory would use the period value as the priority setting for all variables active in that period, or a layered chip manufacturing process where the priority assigned to binary variables is top down or bottom up in that layer.

If priorities are given to binary, integer, or semi-continuous variables, then these are used to provide a user-specified order in which variables are branched. XA selects the variable with the highest priority (lowest numerical value) for branching and the strategy determines the direction, up or down.

Priorities are assigned to variables using the .prior suffix. For example:

```
NAVY.PRIOROPT = 1 ;
...
Z.PRIOR(J,"SMALL") = 10 ;
Z.PRIOR(J,"MEDIUM") = 5 ;
Z.PRIOR(J,"LARGE") = 1 ;
```

The value 1 indicates the highest priority (branch first), and the value 10 the lowest priority (branch last). Valid priority values should range between -32000 and 32000. The default priority value is 16000.

5.2 Branching Strategies

Ten branch & bound strategies are provided to meet the demands of many different types of problems. Each strategy has five variations (six if you include the basic strategy, or “no variation) that affect the solve time, speed to first solution, and the search for the best integer solution. The order in which integer variables are processed during the search is important. This order is called the branching order of integer variables. Solution times can vary significantly with the method selected.

In general, XA will solve your MIP problems much faster when all model variables are given some kind of objective function value. This biases the basis in a specific direction and usually leads to satisfactory first integer solutions.

The strategy used can be changed by setting the "strategy" option in an XA options file.

<table>
<thead>
<tr>
<th>Branch &amp; Bound Strategy</th>
<th>Description of Selection Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Proprietary method. Default value. Excellent strategy, also add priority to integer variable and try 1P for additional performance gains.</td>
</tr>
<tr>
<td>2</td>
<td>Minimum change in the objective function. This strategy has not been very successful at solving MIP problems.</td>
</tr>
<tr>
<td>3</td>
<td>Priority based upon column order. This strategy probably does not have much meaning because you typically do not set the column order in GAMS</td>
</tr>
<tr>
<td>4</td>
<td>Column closest to its integer bound. This strategy tends to send a variable to its lower bounds.</td>
</tr>
<tr>
<td>6</td>
<td>Column always branches up (high). Second choice after 1. Excellent choice when your model is a multi-period problem; additional performance gains when priority value are equated with period number; also try 6P if using priorities.</td>
</tr>
<tr>
<td>7</td>
<td>Column always branches down (low). Useful if variable branched down doesn’t limit capacity or resources. One suggestion is to use priorities in the reverse order from that described in Strategy 6.</td>
</tr>
</tbody>
</table>
Each XA B&B strategy has many variations. Sometimes these variations reduce the solution time but may not yield the optimal integer solution. If you are interested in obtaining a fast and ‘good’ integer solution (which may not be the optimal integer solution), try these variations. You should be aware, though, that using these variations will invalidate the best bound and optimality gap statistics printed by the link at the end of the solve. To choose a variation, either append its letter to the strategy number or add its offset to the strategy number. For example, to choose variations B and P of strategy 6, you could either set "strategy 6BP" or "strategy 1806".

<table>
<thead>
<tr>
<th>Variation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (+100)</td>
<td>This variation reduces the amount of time XA spends estimating the value of a potential integer solution. The values calculated are rough estimates and may eliminate nodes that would lead to better integer solutions. Variation A may not appear with variation B.</td>
</tr>
<tr>
<td>B</td>
<td>This variation spends very little time calculating estimated integer solutions at each node and is the most radical in performance and integer solution value and may eliminate nodes that would lead to better integer solutions. Variation B may not appear with variation A.</td>
</tr>
<tr>
<td>C</td>
<td>Each time an improving integer solution is found XA splits the remaining node list in half based upon the length of the current list. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation C may not appear with variation D.</td>
</tr>
<tr>
<td>D</td>
<td>Each time an improving integer solution is found XA splits the remaining node list based upon the difference in current projected objective and the best possible objective value divided by two. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation D may not appear with variation C.</td>
</tr>
<tr>
<td>P</td>
<td>Each time a node is generated XA calculates the effects of each non-integer on future objective function values, which is calculation intensive. By assigning branching priorities to your integer variables XA will only perform this calculation on the non-integer variables with the lowest branching priority. This frequently reduces the number of calculations. Variation P may appear with any variation, but to be effective you must assign integer branching priorities.</td>
</tr>
</tbody>
</table>

If you wish to improve your solution times, you should experiment with different Strategies to determine which is best for your problems. We have found that Strategies 1 and 6 work quite well. Also try strategies 1A, 1B, 6A, 6B, and 9. As you gain experience with these Strategies you will be able to make an informed choice.

### 5.3 Limitsearch Parameter

LIMITSEARCH is used to limit the number of nodes to search by implicitly or explicitly stating a bound on the value of an integer solution. The integer solution obtained, if any, will have a functional value no worse than LIMITSEARCH. The next integer solution will have a monotonically improving objective function value until an optimal integer solution is found and if verified.

If you can estimate the objective function value of a good integer solution, you can avoid nodes that lead to worse solutions and, consequently, speed up the search. However, too restrictive a value may lead to no integer solution at all, if an integer solution with an objective value better than the LIMITSEARCH value does not exist. If the search terminates with ‘NO INTEGER SOLUTION’, you must begin the search again with a less restrictive LIMITSEARCH value. The LIMITSEARCH command line parameter has three methods of specifying a lower limit on the objective function.
### LIMITSEARCH Value | Meaning
---|---
`##` | Only search for integer solutions between this value and the 'optimal continuous' solution.
`##%` | Only search for integer solutions with `##%` of the 'optimal continuous' solution.
`(##%)` | Solve for the integer solution that is within `(##%)` of the 'optimal integer solution'. This can reduce the search time significantly, but the reported integer solution may not be the optimal integer solution: it will only be within `(##%)` of it. This is similar to the GAMS `optcr` option, but setting `optcr` reports the actual gap: this is the recommended way to run GAMS/XA.

## 6 The XA Option File

The option file is called `xa.opt`. The GAMS model should contain the following line to signal GAMS/XA to use the option file:

```gams
<modelname>.optfile = 1 ;
```

where `<modelname>` is the name of the model specified in the model statement. For instance:

```gams
model m /all/ ;
m.optfile = 1 ;
option LP = XA ;
solve m using LP minimize z ;
```

The XA option file allows you to solver-specific options that are not anticipated by GAMS. Where an XA option and a GAMS option both set the same thing, the setting in the XA option file takes precedence. Option file lines beginning with an asterisk `∗` are comment lines. For example:

- `∗ Integer solving strategy.
  Strategy 6P`
- `∗ Write log information to the screen every 5 seconds.
  Set FreqLog 00:05`
- `∗ Do NOT scale the problem.
  Set Scale No`

The contents of the option file are echoed to the screen. If no options file is found where one is expected, a warning is sent to the log file and the solve continues.

Here is a list of available XA options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>basis</td>
<td>controls writing of advanced basis file for restarts After XA has solved your problem, the solution is saved for the next time the problem is solved. This can greatly reduce the number of iterations and execution time required. The Dual Simplex algorithm is used when XA detects advance basis restarts. You can instruct XA to not use the Dual Simplex algorithm for restarts as follows, set <code>DualSimplex</code> to No.</td>
<td></td>
</tr>
<tr>
<td>force</td>
<td>force internal relaxation of bounds if infeasible If your LP model is infeasible, XA makes adjustments in column and row bounds to make the problem feasible. No adjustments are made to binary columns or RHS values of SOS sets. Depending upon how tightly constrained the problem is, XA may be prevented from making additional adjustment that would lead to an integer solution. No adjustments are made to make a column’s lower bound less than zero.</td>
<td>0</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>limitsearch</td>
<td>limit searching of the branch-and-bound tree</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Section Limitsearch Parameter.</td>
<td></td>
</tr>
<tr>
<td>matlist</td>
<td>display the problem for debugging</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Problem is displayed in equation format. This is probably the most useful command when debugging the model. The GAMS equation and variable listings perform a similar function.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Var List columns in each row</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Con List rows in each column</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Both Do both Var and Con</td>
<td></td>
</tr>
<tr>
<td></td>
<td>None No debugging display</td>
<td></td>
</tr>
<tr>
<td>set_barrier</td>
<td>control the primal-dual interior point algorithm</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Activates XA's primal-dual interior point algorithm. Useful when solving very largescale LP models.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Yes Use barrier, with crossover on MIP only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No Use simplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X Use barrier, with crossover</td>
<td></td>
</tr>
<tr>
<td>set_bell</td>
<td>controls the termination bell</td>
<td>0</td>
</tr>
<tr>
<td>set_bvpriority</td>
<td>set default priority of all binary variables</td>
<td>1600</td>
</tr>
<tr>
<td></td>
<td>By default, all variables have priority 1600, so a value &lt; 1600 causes binary variables to be branched on before general integer variables. A value &gt; 1600 has the opposite affect.</td>
<td></td>
</tr>
<tr>
<td>set_crash</td>
<td>control basis-crashing method</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 Minimize primal infeasibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Minimize dual infeasibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 Both 0 &amp; 1</td>
<td></td>
</tr>
<tr>
<td>set_degeniter</td>
<td>consecutive degenerate pivots before anti-cycling code is activated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Degenerate anticycling aide. Number of consecutive degenerate pivots before anticycling code is activated.</td>
<td></td>
</tr>
<tr>
<td>set_freqlog</td>
<td>Frequency in time to print the iteration log line. A negative number (e.g., -00:02) overwrites the same line. This command reduces the overhead of printing too many iteration lines.</td>
<td></td>
</tr>
<tr>
<td>set_dupsimplex</td>
<td>controls use of dual simplex method on restarts</td>
<td>1</td>
</tr>
<tr>
<td>set_intgaps</td>
<td>time frequency for printing the iteration log</td>
<td></td>
</tr>
<tr>
<td>set_intlimit</td>
<td>limit number of incumbents found</td>
<td></td>
</tr>
<tr>
<td></td>
<td>After finding # improving integer solutions, XA terminates with the best solution found thus far. Reported integer solution may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>set_intpct</td>
<td>% of available integer columns to consider fixing at each node</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Useful on very large binary problems. If 100 is entered then all integer columns that are integer at the end of solving the relaxed LP problem are fixed at the current integer bounds.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 0.0 - meaning no fixing.</td>
<td></td>
</tr>
<tr>
<td>set_iround</td>
<td>control rounding of integer vars when XA reports solutions</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>XA reports either rounded or unrounded integer column primal activity.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>set_iteration</td>
<td>sets limit on simplex iterations. XA terminates if limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Reported integer solution may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>set_ltolerance</td>
<td>lower tolerance for integrality: ((i-\text{utol},i+\text{ltol})) is integral. The tolerances XA uses to decide that an integer column's value is integral. For instance, you might consider using a UTOLERANCE of 0.02 (a boat 98% full is for all practical purposes really 100% full). But beware, these integer activities within the specified tolerances are used in calculating constraint relationships and the objective function value. For example, if LTOLERANCE = 0.001, UTOLERANCE = 0.05, and Y has a reported (rounded) activity of 4.0, then (3 \times Y) is in the range ([3 \times 3.95, 3 \times 4.001]).</td>
<td>5e-6</td>
</tr>
<tr>
<td>set_markowitz</td>
<td>larger values favor sparsity over numeric stability.</td>
<td>10</td>
</tr>
<tr>
<td>set_maxcpu</td>
<td>limits the number of processors to use. In general, MIP models should solve # times faster than on a single processor machine. Consider requesting (50%) more memory per processor. Defaults to the number of processors on the machine. This number can be greater than the number of physical processors.</td>
<td></td>
</tr>
<tr>
<td>set_maxnodes</td>
<td>sets the estimate of the number of nodes in the branch-and-bound tree. Default value: 4,000 plus the number of binary variables plus square root of the number of integer columns. Default value: unlimited.</td>
<td></td>
</tr>
<tr>
<td>set_nodelimit</td>
<td>limits the number of nodes in the branch-and-bound tree.</td>
<td></td>
</tr>
<tr>
<td>set_perturbate</td>
<td>controls perturbation on highly degenerate problems. A positive value allows XA to generate a uniformly distributive random variable between 0 and #. A negative value uses a constant perturbation of the absolute value of #. Note: This option should not be used except when all else fails. XA has build-in routines to handle degeneracy.</td>
<td></td>
</tr>
<tr>
<td>set_pricing</td>
<td>controls the variable pricing strategy. Variable pricing strategies, useful if XA appears to make a substantial number (rows/2) of pivots that do not move the objective function or reduce the sum of infeasibilities. This feature requires more memory because an additional set of matrix coefficient are loaded into memory. 0 Standard reduced cost pricing 1 Automatic DEVEX pricing switch over 2 Infeasible DEVEX pricing 3 Feasible DEVEX pricing (our next choice) 4 Both infeasible and feasible DEVEX pricing</td>
<td>0</td>
</tr>
<tr>
<td>set_reducedcost</td>
<td>zero tolerance for the primal marginals.</td>
<td>1e-7</td>
</tr>
<tr>
<td>set_reinvertfreq</td>
<td>set basis re-inversion frequency. The basis update factors are thrown away and the basis reinverted with this frequency.</td>
<td>40</td>
</tr>
<tr>
<td>set_relaxed</td>
<td>relax integrality requirement in MIP models - solve as an RMIP</td>
<td>0</td>
</tr>
<tr>
<td>set_scale</td>
<td>set scaling method</td>
<td></td>
</tr>
<tr>
<td>set_stickwithit</td>
<td>controls how XA follows branching advice from an integer basis. If an integer basis (.b01) file is reloaded to indicate branching direction for the current XA solve, this branching advice is following until # infeasible branches are made. After # infeasible branches, the standard branching direction for the particular branch &amp; bound strategy is used.</td>
<td>10</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>set_tctolerance</td>
<td>zero tolerance for technology coefficients. The smallest technological coefficient allowed in your matrix array. This tolerance is useful when extensive calculations are performed on these coefficients, where the results should be zero (but because of rounding errors) ends up being something like $1.0 \times 10^{-15}$.</td>
<td>$1 \times 10^{-7}$</td>
</tr>
<tr>
<td>set_timelimit</td>
<td>set the time limit. Maximum time allowed to solving the problem. XA terminates if this limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Units are wall clock time. If set too low, reported integer solutions may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>set_tolerance_dual</td>
<td>dual feasibility tolerance</td>
<td>$1 \times 10^{-7}$</td>
</tr>
<tr>
<td>set_tolerance_primal</td>
<td>primal feasibility tolerance</td>
<td>$1 \times 10^{-7}$</td>
</tr>
<tr>
<td>set_utolerance</td>
<td>upper tolerance for integrality: $(i-\text{utol}, i+\text{ltol})$ is integral. The tolerances XA uses to decide that an integer column's value is integral. For instance, you might consider using a \text{UTOLERANCE} of 0.02 (a boat 98% full is for all practical purposes really 100% full). But beware, these integer activities within the specified tolerances are used in calculating constraint relationships and the objective function value. For example, if \text{LTOLERANCE} = 0.001, \text{UTOLERANCE} = 0.05, and \text{Y} has a reported (rounded) activity of 4.0, then $3 \times \text{Y}$ is in the range $[3 \times 3.95, 3 \times 4.001]$.</td>
<td>$5 \times 10^{-6}$</td>
</tr>
<tr>
<td>set_ypivot</td>
<td>sets a pivot tolerance. When selecting a column to leave the basis, columns with absolute marginal values less than $\text{ypivot}$ are rejected. Pivoting in columns with very small values can lead to numeric instability and should be avoided when possible. Setting $\text{ypivot}$ too large can lead to infeasible pivoting. Extreme caution should be exercised when changing this value because of the overall effect on problem feasibility.</td>
<td>$1 \times 10^{-9}$</td>
</tr>
<tr>
<td>stopafter</td>
<td>post-integer-feasible time limit for MIP models. Amount of time (hh:mm:ss) to continue solving after finding the first integer solution. Reported integer solution may not be the optimal integer solution because of premature termination.</td>
<td></td>
</tr>
<tr>
<td>strategy</td>
<td>controls the MIP search strategy</td>
<td></td>
</tr>
<tr>
<td>tomps</td>
<td>write an MPS file (gams.mps) for the problem \text{gams.mps} file is created or rewritten. \text{Yes}: Write problem in MPS format \text{No}: Do not write an MPS formatted file \text{Secure}: Write problem in MPS format with names C0, C1, ... and R0, R1, ...</td>
<td>\text{No}</td>
</tr>
</tbody>
</table>

### 7 Iteration Log Formats

The iteration log is something many users watch closely, especially when solving MIP models. Setting MUTE YES or Set FreqLog 0 suppresses the display of the iteration log. During LP iterations, the log format varies depending on the algorithm chosen. Its format is self-explanatory. The default MIP log looks like:

```
Node     IInf  ToGo.Map  Best.Obj  Cur.Obj  Int.Obj  # Column +/- Iter
    #### #    ############ #    ############ #    #  #  #  #
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Active node, the smaller the better, value increases and decreases as the branch- and-bound proceeds.</td>
</tr>
<tr>
<td>IInf</td>
<td>Number of discrete columns having fractional values. This number converges to 0 as XA approaches an integer solution.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ToGo.Map</td>
<td>A numeric picture of open nodes. The i’th digit (from the right) represents the number of open nodes in the i’th group of ten nodes. For example, 435 means: - 4 unexplored nodes between nodes 20 and 29. - 3 unexplored nodes between nodes 10 and 19. - 5 unexplored nodes between nodes 0 and 9.</td>
</tr>
<tr>
<td>Best.Obj</td>
<td>Best possible integer objective function. As the branch-and-bound algorithm proceeds this number bounds the Optimal Integer Solution. This number does not change very fast.</td>
</tr>
<tr>
<td>Cur.Obj</td>
<td>Objective function for the current node. If an integer solution is found in this node cannot be any better than this value.</td>
</tr>
<tr>
<td>Int.Obj</td>
<td>Objective function of the best integer solution found so far. This value improves as additional integer solutions are found.</td>
</tr>
<tr>
<td>#</td>
<td>Number of improving integer solutions found thus far.</td>
</tr>
<tr>
<td>Column</td>
<td>Column selected by the branch-and-bound process.</td>
</tr>
<tr>
<td>+/-</td>
<td>Branching direction: up(+) or down(-).</td>
</tr>
<tr>
<td>Iter</td>
<td>Cumulative total of simplex iterations used (including the relaxed LP).</td>
</tr>
</tbody>
</table>

Display of the iteration log line may be toggled on and off by entering a CTRL/U during the iteration process. Use the Set FreqLog command line parameter to minimize the number of lines displayed. Logging each iteration can significantly slow down the solution process.
1 Introduction

This document describes the GAMS/XPRESS linear and mixed-integer programming solver. The GAMS/XPRESS solver is based on the XPRESS-MP Optimization Subroutine Library, and runs only in conjunction with the GAMS modeling system.

GAMS/XPRESS (also simply referred to as XPRESS) is a versatile, high-performance optimization system. The system integrates a powerful simplex-based LP solver, a MIP module with cut generation for integer programming problems and a barrier module implementing a state-of-the-art interior point algorithm for very large LP problems.

The GAMS/XPRESS solver is installed automatically with your GAMS system. Without a license, it will run in student or demonstration mode (i.e. it will solve small models only). If your GAMS license includes XPRESS, there is no size or algorithm restriction imposed by the license, nor is any separate licensing procedure required.

In addition if you have a callable library license from XPRESS, we offer a GAMS/XPRESS link at a reduced price. This is the same program as the GAMS/XPRESS solver, but it uses your existing XPRESS callable library license. It is noted that the link does not work on the XPRESS stand alone license. Finally, a bare-bone interface to the LP and MIP solver of Xpress is available under the name OSIXPRESS. It comes free of charge with any GAMS system.

2 Usage

If you have installed the system and configured XPRESS as the default LP, RMIP\(^1\) and MIP solver, all LP, RMIP and MIP models without a specific solver option will use XPRESS. If you installed another solver as the default, you can explicitly

\(^1\)RMIP means: Relaxed Mixed Integer Programming. You can solve a MIP model as an RMIP. This will ignore the integer restrictions and thus solves the problem as an LP.
request a particular model to be solved by XPRESS by inserting the statement

```plaintext
option LP = xpress; { or MIP or RMIP }
```

somewhere before the solve statement.

The standard GAMS options can be used to control XPRESS-MP. The rules for using an option file are described in section GAMS Options.

XPRESS-specific options can be specified in a file called `xpress.opt`. The syntax is rather simple: a line in the option file can be one of the following:

- An empty line or a line consisting only of blanks.
- A comment line, which is a line in which the first non-blank character is an asterisk `*`. The remainder of the line is ignored.
- An option, which consists of a keyword followed by a value.

An example of a valid option file is:

```plaintext
* sample XPRESS-MP options file
algorithm simplex
 presolve 0
 IterLim 50000
```

Keywords are not case sensitive. I.e. whether you specify `iterlim`, `ITERLIM`, or `Iterlim` the same option is set. To use an options file you specify a model suffix `modelname.optfile=1;` or use command line options `optfile=1`. The rules for using solver-specific options are also described in section The Solver Option File.

In general this is enough knowledge to solve your models. In some cases you may want to use some of the XPRESS options to gain further performance improvements or for other reasons.

### 2.1 General

The options `advBasis`, `algorithm`, `basisOut`, `mpsOutputFile`, `reform`, `reRun`, and `reslim` control the behavior of the GAMS/XPRESS link. The options `crash`, `extraPresolve`, `lpIterlimit`, `presolve`, `scaling`, `threads`, and `trace` set XPRESS library control variables, and can be used to fine-tune XPRESS. See section General Options for more details of XPRESS general options.

### 2.2 LP

See section LP Options for more details of XPRESS library control variables which can be used to fine-tune the XPRESS LP solver.

### 2.3 MIP

In some cases, the branch-and-bound MIP algorithm will stop with a proven optimal solution or when unboundedness or (integer) infeasibility is detected. In most cases, however, the global search is stopped through one of the generic GAMS options:

1. `iterlim` (on the cumulative pivot count), `reslim` (in seconds of CPU time),
2. `optca` & `optcr` (stopping criteria based on gap between best integer solution found and best possible) or
3. `nodlim` (on the total number of nodes allowed in the B&B tree).
It is also possible to set the `maxNode` and `maxMipSol` options to stop the global search: see section MIP Options for XPRESS control variables for MIP. The options `loadMipSol`, `mipCleanup`, `mipTrace`, `mipTraceNode`, and `mipTraceTime` control the behavior of the GAMS/XPRESS link on MIP models. The other options in section MIP Options set XPRESS library control variables, and can be used to fine-tune the XPRESS MIP solver.

### 2.4 MIP Solution Pool

Typically, XPRESS finds a number of integer feasible points during its global search, but only the final solution is available. The MIP solution pool capability makes it possible to store multiple integer feasible points (aka solutions) for later processing. The MIP solution pool operates in one of two modes: by default (solnpoolPop = 1) the global search is not altered, but with (solnpoolPop = 2) a selected set (potentially all) of the integer feasible solutions are enumerated.

The MIP enumeration proceeds until all MIP solutions are enumerated or cut off, or until a user-defined limit is reached. Whenever a new solution is generated by the enumerator, it is presented to the solution pool manager. If there is room in the pool, the new solution is added. If the pool is full, a `cull round` is performed to select a number of solutions to be thrown out - these solutions can be those stored in the pool and/or the new solution. Solutions can be selected for culling based on their MIP objective value and/or the overall diversity of the solutions in the pool. If neither is chosen, a default choice is made to throw out one solution based on objective values. Whenever a solution is thrown out based on its MIP objective, the enumeration space is pruned based on the cutoff defined by this objective value.

By default, the capacity of the pool is set very large, as is the number of cull rounds to perform, so selecting only solnpoolPop = 2 will result in full enumeration. However, many different strategies can be executed by setting the solution pool options. For example, to choose the N-best solutions, simply set the solution pool capacity to N. When the pool is full, new solutions will force a cull round, and the default is to reject one solution based on its objective and update the cutoff accordingly. To generate all solutions with an objective as good as X, leave the pool capacity set at a high level but set the cutoff to X using the `mipabscutoff` option. To return the N-first solutions, set the solution pool capacity to N and solnpool1CullRounds = 0: as soon as the pool is full the enumeration will stop on the cull round limit.

A number of other strategies for controlling the solution pool behavior are possible by combining different options. Several working examples are provided in the GAMS Test Library in models xpress03.gms, xpress04.gms, and xpress05.gms. See section MIP Solution Pool Options for XPRESS control variables for MIP Solution Pool.

### 2.5 Newton-Barrier

The barrier method is invoked by default for quadratic problems, and can be selected for linear models by using one of the options

```
algorithm          barrier
defaultalg         4
```

The barrier method is likely to use more memory than the simplex method. No warm start is done, so if an advanced basis exists, you may not wish to use the barrier solver.

See section Newton-barrier Options for XPRESS control variables for Newton-Barrier method.

### 3 Options

The tables that follow contain the XPRESS options. They are organized by function (e.g. LP or MIP) and also by type: some options control the behavior of the GAMS/XPRESS link and will be new even to experienced XPRESS users, while other options exist merely to set control variables in the XPRESS library and may be familiar to XPRESS users.

#### 3.1 General Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>advBasis</td>
<td>use advanced basis provided by GAMS</td>
<td></td>
</tr>
<tr>
<td>algorithm</td>
<td>choose between simplex and barrier algorithm</td>
<td>simplex</td>
</tr>
<tr>
<td></td>
<td>This option is used to select the barrier method to solve LPs. By default the barrier method will do a crossover to find a basic solution.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>barrier Use the barrier algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>simplex Use the simplex algorithm</td>
<td></td>
</tr>
<tr>
<td>basisOut</td>
<td>directs optimizer to output an MPS basis file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>In general this option is not used in a GAMS environment, as GAMS maintains basis information for you automatically.</td>
<td></td>
</tr>
<tr>
<td>crash</td>
<td>control for basis crashing procedure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A crash procedure is used to quickly find a good basis. This option is only relevant when no advanced basis is available.</td>
<td></td>
</tr>
<tr>
<td>deterministic</td>
<td>control for deterministic behavior of concurrent solves</td>
<td>1</td>
</tr>
<tr>
<td>extraPresolve</td>
<td>initial number of extra elements to allow for in the presolve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The space required to store extra presolve elements is allocated dynamically, so it is not necessary to set this control. In some cases, the presolve may terminate early if this is not increased.</td>
<td></td>
</tr>
<tr>
<td>lpIterLimit</td>
<td>set the iteration limit for simplex solves</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For MIP models, this is a per-node iteration limit for the B&amp;B tree. Overrides the iterlim option.</td>
<td></td>
</tr>
<tr>
<td>mpsNameLength</td>
<td>maximum number of integer solutions in MIP tree search</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Maximum length of MPS names in characters. Internally it is rounded up to the smallest multiple of 8. MPS names are right padded with blanks. Maximum value is 64.</td>
<td></td>
</tr>
<tr>
<td>mpsOutputFile</td>
<td>Name of MPS output file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If specified XPRESS-MP will generate an MPS file corresponding to the GAMS model. The argument is the file name to be used. It can not have an extension: XPRESS-MP forces the extension to be .MAT even if an extension was specified. You can prefix the file name with a path.</td>
<td></td>
</tr>
<tr>
<td>presolve</td>
<td>sets presolve strategy</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 presolve not applied</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 presolve applied</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 presolve applied, but redundant bounds are not removed</td>
<td></td>
</tr>
<tr>
<td>reform</td>
<td>substitute out objective var and equ when possible</td>
<td>1</td>
</tr>
<tr>
<td>reRun</td>
<td>rerun with primal simplex when not optimal/feasible</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Applies only in cases where presolve is turned on and the model is diagnosed as infeasible or unbounded. If rerun is nonzero, we rerun the model using primal simplex with presolve turned off in hopes of getting better diagnostic information. If rerun is zero, no good diagnostic information exists, so we return no solution, only an indication of unboundedness/infeasibility.</td>
<td></td>
</tr>
<tr>
<td>reslim</td>
<td>overrides GAMS reslim option</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sets the resource limit. When the solver has used more than this amount of CPU time (in seconds) the system will stop the search and report the best solution found so far.</td>
<td></td>
</tr>
<tr>
<td>scaling</td>
<td>bitmap control for internal scaling algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bitmap to determine how internal scaling is done. If set to 0, no scaling will take place. The default of 35 implies row and column scaling done by the maximum element method.</td>
<td></td>
</tr>
<tr>
<td>threads</td>
<td>global default thread count</td>
<td>1</td>
</tr>
<tr>
<td>trace</td>
<td>turns on output of infeasibility diagnosis during presolve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Control of the infeasibility diagnosis during presolve - if nonzero, infeasibility will be explained.</td>
<td></td>
</tr>
<tr>
<td>writePrtSol</td>
<td>directs optimizer to output a &quot;printsol&quot; file</td>
<td></td>
</tr>
</tbody>
</table>
### 3.2 LP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigM</td>
<td>infeasibility penalty used in the ''big M'' method</td>
<td>0</td>
</tr>
<tr>
<td>bigMMethod</td>
<td>controls use of ''big M'' method - 0=no, 1=yes</td>
<td></td>
</tr>
<tr>
<td>concurrentThreads</td>
<td>control for concurrent LP algorithm</td>
<td></td>
</tr>
<tr>
<td>defaultAlg</td>
<td>sets the default LP algorithm</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1 automatic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 dual simplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 primal simplex</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 Newton barrier</td>
<td></td>
</tr>
<tr>
<td>eigenvalueTol</td>
<td>zero tolerance for negative eigenvalues of quadratic matrices</td>
<td>1e-6</td>
</tr>
<tr>
<td>etaTol</td>
<td>zero tolerance on eta elements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>During each iteration, the basis inverse is premultiplied by an elementary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>matrix, which is the identity except for one column the eta vector. Elements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of eta vectors whose absolute value is smaller than ( \text{etaTol} ) are</td>
<td></td>
</tr>
<tr>
<td></td>
<td>taken to be zero in this step.</td>
<td></td>
</tr>
<tr>
<td>feasTol</td>
<td>zero tolerance for RHS and bound values</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>This is the zero tolerance on right hand side values, bounds and range values.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If one of these is less than or equal to feasTol in absolute value, it is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>treated as zero.</td>
<td></td>
</tr>
<tr>
<td>invertFreq</td>
<td>frequency of basis re-inversion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The frequency with which the basis will be inverted. A value of -1 implies</td>
<td></td>
</tr>
<tr>
<td></td>
<td>automatic.</td>
<td></td>
</tr>
<tr>
<td>invertMin</td>
<td>minimum number of iterations between basis re-inversion</td>
<td></td>
</tr>
<tr>
<td>lpLog</td>
<td>print control for LP log</td>
<td></td>
</tr>
<tr>
<td>lpThreads</td>
<td>control for concurrent LP algorithm: alias for concurrentThreads</td>
<td></td>
</tr>
<tr>
<td>matrixTol</td>
<td>zero tolerance on matrix elements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The zero tolerance on matrix elements. If the value of a matrix element is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>less than or equal to matrixTol in absolute value, it is treated as zero.</td>
<td></td>
</tr>
<tr>
<td>optimalityTol</td>
<td>zero tolerance on reduced costs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>On each iteration, the simplex method searches for a variable to enter the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basis which has a negative reduced cost. The candidates are only those</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variables which have reduced costs less than the negative value of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>optimalityTol</td>
<td></td>
</tr>
<tr>
<td>penalty</td>
<td>minimum absolute penalty variable coefficient used in the ''big M'' method</td>
<td></td>
</tr>
<tr>
<td>pivotTol</td>
<td>zero tolerance on pivot elements in simplex method</td>
<td></td>
</tr>
<tr>
<td></td>
<td>On each iteration, the simplex method seeks a nonzero matrix element to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pivot on. Any element with absolute value less than pivotTol is treated as</td>
<td></td>
</tr>
<tr>
<td></td>
<td>zero for this purpose</td>
<td></td>
</tr>
<tr>
<td>pricingAlg</td>
<td>determines the pricing method to use</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This determines the pricing method to use on each iteration, selecting which</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variable enters the basis. In general Devex pricing requires more time on</td>
<td></td>
</tr>
<tr>
<td></td>
<td>each iteration, but may reduce the total number of iterations, whereas</td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial pricing saves time on each iteration, although possibly results in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>more iterations.</td>
<td></td>
</tr>
<tr>
<td>relPivotTol</td>
<td>minimum size of pivot element relative to largest element in column</td>
<td></td>
</tr>
<tr>
<td></td>
<td>At each iteration a pivot element is chosen within a given column of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>matrix. The relative pivot tolerance, relpivotTol, is the size of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element chosen relative to the largest possible pivot element in the same</td>
<td></td>
</tr>
<tr>
<td></td>
<td>column.</td>
<td></td>
</tr>
</tbody>
</table>

### 3.3 MIP Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>backTrack</td>
<td>determines selection of next node in tree search</td>
<td>1</td>
</tr>
<tr>
<td>breadthFirst</td>
<td>determines number of nodes to include in a breadth-first search. If <code>nodeselection = 4</code>, this determines the number of nodes to include in a breadth-first search.</td>
<td>1</td>
</tr>
<tr>
<td>coverCuts</td>
<td>number of rounds of lifted cover inequalities at the top node. A lifted cover inequality is an additional constraint that can be particularly effective at reducing the size of the feasible region without removing potential integral solutions. The process of generating these can be carried out a number of times, further reducing the feasible region, albeit incurring a time penalty. There is usually a good payoff from generating these at the top node, since these inequalities then apply to every subsequent node in the tree search.</td>
<td></td>
</tr>
<tr>
<td>cutDepth</td>
<td>maximum depth in search tree at which cuts will be generated. Generating cuts can take a lot of time, and is often less important at deeper levels of the tree since tighter bounds on the variables have already reduced the feasible region. A value of 0 signifies that no cuts will be generated.</td>
<td></td>
</tr>
<tr>
<td>cutFreq</td>
<td>frequency at which cuts are generated in the tree search. If the depth of the node modulo <code>cutfreq</code> is zero, then cuts will be generated.</td>
<td></td>
</tr>
<tr>
<td>cutStrategy</td>
<td>specifies the cut strategy. An aggressive cut strategy, generating a greater number of cuts, will result in fewer nodes to be explored, but with an associated time cost in generating the cuts. The fewer cuts generated, the less time taken, but the greater subsequent number of nodes to be explored.</td>
<td></td>
</tr>
<tr>
<td>gomCuts</td>
<td>number of rounds of Gomory cuts at the top node. These can always be generated if the current node does not yield an integral solution. However, Gomory cuts are not usually as effective as lifted cover inequalities in reducing the size of the feasible region.</td>
<td></td>
</tr>
<tr>
<td>heurThreads</td>
<td>number of threads for running root node heuristics</td>
<td>0</td>
</tr>
<tr>
<td>loadMipSol</td>
<td>loads a MIP solution (the initial point)</td>
<td>0</td>
</tr>
<tr>
<td>maxMipSol</td>
<td>maximum number of integer solutions in MIP tree search. This specifies a limit on the number of integer solutions to be found (the total number, not necessarily the number of distinct solutions). 0 means no limit.</td>
<td></td>
</tr>
<tr>
<td>maxNode</td>
<td>maximum number of nodes to explore in MIP tree search. If the GAMS nodlim model suffix is set, that setting takes precedence.</td>
<td></td>
</tr>
<tr>
<td>mipAbsCutoff</td>
<td>nodes with objective worse than this value are ignored. If the user knows that they are interested only in values of the objective function which are better than some value, this can be assigned to <code>mipabscutoff</code>. This allows the Optimizer to ignore solving any nodes which may yield worse objective values, saving solution time.</td>
<td></td>
</tr>
<tr>
<td>mipAbsStop</td>
<td>stopping tolerance for gap: if met XPRESS returns proven optimal. The global search is stopped if the gap is reduced to this value. This check is implemented in the Optimizer library, and if the search is stopped on this check the Optimizer returns a status of proven optimal. For this reason you should use the GAMS <code>&lt;modelname&gt;.optca</code> parameter instead of this option.</td>
<td></td>
</tr>
<tr>
<td><strong>mipAddCutoff</strong></td>
<td>amount to add to MIP incumbent to get the new cutoff</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The amount to add to the objective function of the best integer solution found to give the new cutoff. Once an integer solution has been found whose objective function is equal to or better than mipabscutoff, improvements on this value may not be interesting unless they are better by at least a certain amount. If mipaddcutoff is nonzero, it will be added to mipabscutoff each time an integer solution is found which is better than this new value. This cuts off sections of the tree whose solutions would not represent substantial improvements in the objective function, saving processor time. Note that this should usually be set to a negative number for minimization problems, and positive for maximization problems. Notice further that the maximum of the absolute and relative cut is actually used.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipCleanup</strong></th>
<th>clean up the MIP solution (round-fix-solve) to get duals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If nonzero, clean up the integer solution obtained, i.e. round and fix the discrete variables and re-solve as an LP to get some marginal values for the discrete vars.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipLog</strong></th>
<th>print control for MIP log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipPresolve</strong></th>
<th>bitmap controlling the MIP presolve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bitmap determining type of integer processing to be performed. If set to 0, no processing will be performed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipRelCutoff</strong></th>
<th>relative difference between the MIP incumbent and the new cutoff</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Percentage of the LP solution value to be added to the value of the objective function when an integer solution is found, to give the new value of mipabscutoff. The effect is to cut off the search in parts of the tree whose best possible objective function would not be substantially better than the current solution.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipRelStop</strong></th>
<th>stopping tolerance for relative gap: if met XPRESS returns proven optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The global search is stopped if the relative gap is reduced to this value. This check is implemented in the Optimizer library, and if the search is stopped on this check the Optimizer returns a status of proven optimal. For this reason you should use the GAMS <code>&lt;modelname&gt;.optcr</code> parameter instead of this option.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipThreads</strong></th>
<th>number of threads for parallel mip algorithm</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>mipTol</strong></th>
<th>integrality tolerance for discrete vars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This is the tolerance within which a decision variables value is considered to be integral.</td>
</tr>
<tr>
<td></td>
<td>5e-6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipTrace</strong></th>
<th>name of MIP trace file</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A miptrace file with the specified name will be created. This file records the best integer and best bound values every miptracenode nodes and at miptraceTime-second intervals.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipTraceNode</strong></th>
<th>node interval between MIP trace file entries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mipTraceTime</strong></th>
<th>time interval, in seconds, between MIP trace file entries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>nodeSelection</strong></th>
<th>sets node selection strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>objGoodEnough</strong></th>
<th>stop once an objective this good is found</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>preprobing</strong></th>
<th>set per-node probing strategy for binary variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1 automatic</td>
</tr>
<tr>
<td></td>
<td>0 disabled</td>
</tr>
<tr>
<td></td>
<td>1 light probing</td>
</tr>
<tr>
<td></td>
<td>2 full probing - all binaries examined</td>
</tr>
<tr>
<td></td>
<td>3 full probing and repeat while effective</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>pseudoCost</strong></th>
<th>default pseudo-cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The default pseudo cost used in estimation of the degradation associated with an unexplored node in the tree search. A pseudo cost is associated with each integer decision variable and is an estimate of the amount by which the objective function will be worse if that variable is forced to an integral value.</td>
</tr>
</tbody>
</table>
### 3.4 MIP Solution Pool Options

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>solnpool</td>
<td>solution pool file name</td>
<td></td>
</tr>
<tr>
<td>solnpoolCapacity</td>
<td>limit on number of solutions to store</td>
<td>9999999999</td>
</tr>
<tr>
<td>solnpoolCullDiversity</td>
<td>cull N solutions based on solution diversity</td>
<td>-1</td>
</tr>
<tr>
<td>solnpoolCullObj</td>
<td>cull N solutions based on objective values</td>
<td>-1</td>
</tr>
<tr>
<td>solnpoolCullRounds</td>
<td>terminate enumeration after N culling rounds</td>
<td>9999999999</td>
</tr>
<tr>
<td>solnpoolDupPolicy</td>
<td>sets policy for detecting/storing duplicate solutions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 keep all</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 compare all vars, exact matches discarded</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 compare rounded discrete, exact continuous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 compare rounded discrete only</td>
<td></td>
</tr>
<tr>
<td>solnpoolmerge</td>
<td>solution pool file name for merged solutions</td>
<td></td>
</tr>
<tr>
<td>solnpoolnumsym</td>
<td>maximum number of variable symbols when writing merged solutions</td>
<td>10</td>
</tr>
<tr>
<td>solnpoolPop</td>
<td>controls method used to populate the solution pool</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 generate solutions using the normal search algorithm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 invoke the solution enumerator to generate solutions</td>
<td></td>
</tr>
<tr>
<td>solnpoolPrefix</td>
<td>file name prefix for GDX solution files</td>
<td>soln</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>solnpoolVerbosity</td>
<td>controls verbosity of solution pool routines</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.5 QP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifCheckConvexity</td>
<td>controls convexity check for QP models - 0=no, 1=yes</td>
<td></td>
</tr>
</tbody>
</table>

### 3.6 Newton-barrier Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>barAlg</td>
<td>determines which barrier algorithm to use</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>-1 automatic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 unused</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 infeasible-start barrier alg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 homogeneous self-dual barrier alg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 start with 2 optionally switch to 1</td>
<td></td>
</tr>
<tr>
<td>barCrash</td>
<td>determines the type of crash used for the crossover from barrier</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>barDualStop</td>
<td>stopping tolerance for dual infeasibilities in barrier: 0=auto</td>
<td>0</td>
</tr>
<tr>
<td>barGapStop</td>
<td>stopping tolerance for relative duality gap in barrier: 0=auto</td>
<td>0</td>
</tr>
<tr>
<td>barIndefLimit</td>
<td>limit consecutive indefinite barrier iterations that will be performed</td>
<td>15</td>
</tr>
<tr>
<td>barIterLimit</td>
<td>maximum number of barrier iterations</td>
<td></td>
</tr>
<tr>
<td>barOrder</td>
<td>controls the Cholesky factorization in barrier: 0=auto</td>
<td>0</td>
</tr>
<tr>
<td>barOutput</td>
<td>controls the level of solution output from barrier</td>
<td>1</td>
</tr>
<tr>
<td>barPrimalStop</td>
<td>stopping tolerance for primal infeasibilities in barrier: 0=auto</td>
<td>0</td>
</tr>
<tr>
<td>barStart</td>
<td>controls the computation of the barrier starting point: 0=auto</td>
<td>0</td>
</tr>
<tr>
<td>barStepStop</td>
<td>stopping tolerance on the step size of the barrier search direction</td>
<td>1e-10</td>
</tr>
<tr>
<td>barThreads</td>
<td>number of threads for parallel barrier algorithm</td>
<td></td>
</tr>
<tr>
<td>crossover</td>
<td>crossover control for barrier method Determines whether the barrier method</td>
<td></td>
</tr>
<tr>
<td></td>
<td>will cross over to the simplex method when at optimal solution has been</td>
<td></td>
</tr>
<tr>
<td></td>
<td>found, in order to provide an end basis.</td>
<td></td>
</tr>
</tbody>
</table>

### 4 Helpful Hints

The comments below should help both novice and experienced GAMS users to better understand and make use of GAMS/XPRESS.

- **Infeasible and unbounded models** The fact that a model is infeasible/unbounded can be detected at two stages: during the presolve and during the simplex or barrier algorithm. In the first case we cannot recover a solution, nor is any information regarding the infeasible/unbounded constraint or variable provided (at least in a way that can be returned to GAMS). In such a situation, the GAMS link will automatically rerun the model using primal simplex with presolve turned off (this can be avoided by setting the rerun option to 0). It is possible (but very unlikely) that the simplex method will solve a model to optimality while the presolve claims the model is infeasible/unbounded (due to feasibility tolerances in the simplex and barrier algorithms).

- The barrier method does not make use of iterlim. Use bariterlim in an options file instead. The number of barrier iterations is echoed to the log and listing file. If the barrier iteration limit is reached during the barrier algorithm, XPRESS continues with a simplex algorithm, which will obey the iterlim setting.
• Semi-integer variables are not implemented in the link, nor are they supported by XPRESS; if present, they trigger an error message.

• SOS1 and SOS2 variables are required by XPRESS to have lower bounds of 0 and nonnegative upper bounds.
Part II

The Free Solvers
1 Introduction

GAMS/BDMLP is a LP and MIP solver that comes for free with any GAMS system. It is intended for small to medium sized models. GAMS/BDMLP was originally developed at the World Bank by T. Brooke, A. Drud, and A. Meeraus and is now maintained by GAMS Development Corp. The MIP part was added by M. Bussieck and A. Drud. GAMS/BDMLP runs on all platforms for which GAMS is available.

GAMS/BDMLP can solve reasonably sized LP models as long as the models are not very degenerate and are well scaled. The Branch-and-Bound algorithm for solving MIP is not in the same league as other commercial MIP codes that are hooked up to GAMS. Nevertheless, the MIP part of GAMS/BDMLP provides free access to a MIP solver that supports all of the discrete variable types supported by GAMS: Binary, Integer, Semiint, Semiint, Sos1, Sos2.

2 How to Run a Model with BDMLP

GAMS/BDMLP can solve models of the following types: LP, RMIP, and MIP. If you did not specify BDMLP as the default LP, RMIP, or MIP solver, use the following statement in your GAMS model before the solve statement:

```gams
option lp = bdmlp; { or RMIP or MIP }
```
BENCH

1 Introduction

BENCH is a GAMS solver to help facilitate benchmarking of GAMS optimization solvers. BENCH calls all user-specified GAMS solvers for a particular modeltype and captures results in the standard GAMS listing file format. BENCH can call the GAMS/EXAMINER solver automatically to independently verify feasibility and optimality of the solution returned.

There are several advantages to using the BENCH solver instead of just creating a GAMS model or batch file to do multiple solves with the various solvers. The first is that the model does not need to be generated individually for each solve: BENCH spawns each solver using the matrix file generated during the initial call to BENCH. Furthermore, BENCH simplifies solution examination/verification by automatically utilizing EXAMINER. And finally, data can automatically be collected for use with the PAVER performance analysis server.

BENCH comes free with any GAMS system. Licensing is dependent on licensing of the subsolvers. Thus, BENCH runs all solvers for which the user has a valid license.

1.1 How to run a Model with BENCH:

BENCH is run like any other GAMS solver. From the command line this is:

```
>> gams modelname modeltype=bench
```

where `modelname` is the GAMS model name and `modeltype` the solver indicator for a particular model type (e.g. LP, MIP, RMIP, QCP, MIQCP, RMIQCP, NLP, DNLP, CNS, MINLP, or MCP). BENCH can also be specified via the option statement within the model itself before the solve statement:
option modeltype=bench;

The user must specify the solvers to be included by using the solvers option (specified in a solver option file called bench.opt). Otherwise, GAMS/BENCH returns with a warning message

Warning: no solvers selected. Nothing to be done.

For more information on using solver option files and the solvers option, see Section User-Specified Options.

2 User-Specified Options

2.1 GAMS Options

BENCH works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are nodlim, optca, optcr, optfile, cheat, cutoff, prioropt, and tryint. These options are global in the sense that they are passed on to all subsolvers called.

The options can be set either through an option statement

option optfile=1;

or through a model suffix, which sets them only for an individual model

modelname.optfile=1;

All of the options listed in the Chapter Basic Solver Usage are implemented in BENCH and are passed on to the respective solvers. We remark that for a particular subsolver some of these options may not be valid. In this case, although they are passed on by BENCH to the respective subsolver, they may not be used.

The options listed below differ from the usual implementation and are based on individual limits for each solver called by BENCH.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterlim</td>
<td>Sets the individual iteration limit. The subsolver called by BENCH will terminate and pass on the current solution if the number of iterations for each solver exceeds this limit.</td>
<td>2000000000</td>
</tr>
<tr>
<td>reslim</td>
<td>Sets the individual time limit in seconds. The subsolver called by BENCH will terminate and pass on the current solution if the resource time for each solver exceeds this limit.</td>
<td>1000</td>
</tr>
</tbody>
</table>

2.2 The BENCH Options

BENCH solver options are passed on through solver option files. If you specify <filename>.optfile = 1; before the SOLVE statement in your GAMS model, BENCH will then look for and read an option file with the name bench.opt (see The Solver Option File for general use of solver option files). Unless explicitly specified in the BENCH option file, the solvers called by BENCH will not read option files. The syntax for the BENCH option file is

optname value

with one option on each line.

For example,

solvers conopt.1 minos snopt.2
This option determines the solvers to be called and is required. If the `solvers` option is omitted, then BENCH terminates with a warning message.

In this example, CONOPT will be called first with the option file `conopt.opt`. Then MINOS will be called with no option file and SNOPT will be called last with the option file `snopt.op2`. We note that the solvers are called in this order. This can be of particular use since detailed solution information at the end of the GAMS listing file is for the final solver called. The input of the solver option file is echoed in the listing file created by BENCH to help distinguish the different solver calls. See the section describing the BENCH listing file for details.

Specifying separate solver option files can also be useful to specify the subsolver that is to be used by a solver.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>alllsolvers</td>
<td>Indicator whether all valid solvers for given modeltype should be run.</td>
<td>0</td>
</tr>
<tr>
<td>dualcstol</td>
<td><strong>Tolerance on complementary slackness between dual variables and the primal constraints</strong>: passed to EXAMINER</td>
<td>1e-7</td>
</tr>
<tr>
<td>dualfeastol</td>
<td><strong>Dual variables and constraints feasibility tolerance</strong>: passed to EXAMINER</td>
<td>1e-6</td>
</tr>
<tr>
<td>examiner</td>
<td>Indicator whether to call GAMS/EXAMINER to independently verify solution for feasibility and optimality.</td>
<td>0</td>
</tr>
<tr>
<td>outlev</td>
<td>Log output level. 1 BENCH summary log output only 2 BENCH summary log output and log output of each solver</td>
<td>2</td>
</tr>
<tr>
<td>paver</td>
<td>Indicator whether PAVER trace files should be written. Enabling causes a trace file <code>solver.pvr</code> to be written for each solver called. If the solver uses an option file, then the resulting file is <code>solver-optnum.pvr</code>, where optnum is the option file number. The files created can be submitted to the PAVER Server for automated performance analysis.</td>
<td>0</td>
</tr>
<tr>
<td>paverex</td>
<td>Indicator whether PAVER trace files should be written for the Examiner run. Enabling causes a trace file <code>solver-ex.pvr</code> to be written for each solver called. If any Examiner check fails (independent or solver), then we return a model status 14 (no solution returned) and a solver status of 4 (terminated by solver). If no Examiner check is done, for example, because the return status is infeasible, then the status codes are returned as is. If the solver uses an option file, then the resulting file is <code>solver-optnum-ex.pvr</code>, where optnum is the option file number. The files created can be submitted to the PAVER Server for automated performance analysis.</td>
<td>0</td>
</tr>
<tr>
<td>primalcstol</td>
<td><strong>Tolerance on complementary slackness between primal variables and the dual constraints</strong>: passed to EXAMINER</td>
<td>1e-7</td>
</tr>
<tr>
<td>primalfeastol</td>
<td><strong>Primal variables and constraints feasibility tolerance</strong>: passed to EXAMINER</td>
<td>1e-6</td>
</tr>
<tr>
<td>returnlastsol</td>
<td>Indicator whether to return solution from the last solver</td>
<td>0</td>
</tr>
<tr>
<td>solvers</td>
<td>List of solvers to benchmark. <code>solver[n]</code> gives the name of the GAMS solver that should be used, where n is the integer corresponding to the options file. If .n is missing, the solver will not look for an options file. <strong>This is a required option.</strong></td>
<td></td>
</tr>
</tbody>
</table>

3 Benchmark Analysis Using the PAVER Server

Benchmark data obtained using GAMS/BENCH can be automatically analyzed using the PAVER Server.

In order to enable creation of the necessary data files for submission to PAVER, users must enable the `paver` option.

For example, suppose a user has a set of models and wishes to compare three solvers, say CONOPT3, MINOS, and SNOPT. The user would then create a `bench.opt` solver option file with the entries

```
solvers conopt3 minos snopt
paver 1
```
Solving the models using bench as the solver will create PAVER data files, namely one for each solver: conopt.pvr, minos.pvr, and snopt.pvr, which can be submitted to the PAVER server at

http://www.gamsworld.org/performance/paver/pprocess_submit.htm

for automated analysis. Note that all PAVER trace files are appended to if they exist and if subsequent solves are made.

4 Solution Verification Using Examiner

4.1 Examiner Checks

BENCH can automatically call the GAMS/EXAMINER solver to check the solution for feasibility and complementarity. In particular, EXAMINER checks for

- primal feasibility: feasibility of both primal variables and primal constraints.
- dual feasibility: feasibility of both dual variables and dual constraints.
- primal complementary slackness: complementary slackness of primal variables to dual constraints.
- dual complementary slackness: complementary slackness of dual variables to primal constraints.

where EXAMINER does two types of checks:

- **Solvepoint**: the point returned by the solver. The solver returns both level and marginal values for the rows and columns: Examiner uses these exactly as given.
- **Solupoint**: EXAMINER uses the variable levels (primal variables) and equation marginals (dual variables) to compute the equation levels and variable marginals. The variable levels and equation marginals used are those returned by the subsolver.

By default, BENCH does not call EXAMINER to verify the solution. To enable solution verification, specify

```
examiner 1
```

in the `bench.opt` solver option file. Of interest are also the EXAMINER tolerances dualcstol, dualfeastol, primalcstol, and primalfeastol, which can also be set in the BENCH solver option file. For more information, see the EXAMINER documentation.

4.2 Examiner Output in BENCH

Examiner output, if solution verification is enabled, is given in the log output during the actual solve and summary information is given in the final BENCH summary under the Examiner column. Models either pass (P) or fail (F) based on the default Examiner or user-specified tolerances given. If EXAMINER does not do a check, for example, because the solver returns a model status of infeasible, then the Examiner column is given as (N).

If Examiner is not enabled, then n/a is listed under the Examiner column.

The first entry under Examiner is the Examiner status for using solver provided variable constraint level values (solvepoint). The second entry is the solupoint, where GAMS computes the constraint levels from the variable levels returned by the solver.

An example is given below, where we specified to use the solvers BDMLP, MINOS, XPRESS, and CPLEX on the GAMS Model Library model transport:
In the example below, EXAMINER is enabled, but does not perform any checks because the return status of the solver lists the model as infeasible (see the Examiner column (N/N)).

For models having discrete variables, for example MIP, MIQCP, or MINLP, we also show the best bound. A sample output using the GAMS model library model magic is shown below.

5 Output

5.1 The BENCH Log File

The BENCH log output contains complete log information for each solver called. The individual solver calls are indicated by the entry

--- Spawning solver : (Solver Name)

followed by the log output of the individual solver.

An example of the log output using the transportation model transport) from the GAMS model library. We specify the solvers BDMLP, XPRESS, MINOS and CPLEX via the option file bench.opt:

--- Starting compilation
--- transport.gms(69) 3 Mb
--- Starting execution
--- transport.gms(45) 4 Mb
--- Generating model transport
--- transport.gms(66) 4 Mb
--- 6 rows, 7 columns, and 19 non-zeroes.
--- Executing BENCH

GAMS/BENCH Jan 19, 2004 LNX.00.NA 21.3 004.027.041.LXI

GAMS Benchmark Solver

Reading user supplied options file /home/gams/support/bench.opt
Processing...
> solvers bdmlp minos xpress cplex

--- Spawning solver : BDMLP

BDMLP 1.3 Jan 19, 2004 LNX.00.01 21.3 058.050.041.LXI

Reading data...
Work space allocated -- 0.03 Mb

<table>
<thead>
<tr>
<th>Iter</th>
<th>Sinf/Objective</th>
<th>Status</th>
<th>Num</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.25000000E+02</td>
<td>infeas</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1.53675000E+02</td>
<td>nopt</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

SOLVER STATUS: 1 NORMAL COMPLETION
MODEL STATUS : 1 OPTIMAL
OBJECTIVE VALUE 153.67500

--- Spawning solver : MINOS

MINOS-Link Jan 19, 2004 LNX.MS.MS 21.3 029.050.041.LXI GAMS/MINOS 5.51

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated -- 1.01 Mb

Reading Rows...
Reading Columns...

EXIT - Optimal Solution found, objective: 153.6750

--- Spawning solver : XPRESS

Xpress-MP licensed by Dash to GAMS Development Corp. for GAMS

Reading data . . . done.

Reading Problem gmsxp_xx
Problem Statistics

Problem Statistics
- 6 ( 0 spare) rows
- 7 ( 0 spare) structural columns
- 19 ( 0 spare) non-zero elements

Global Statistics
- 0 entities 0 sets 0 set members
- Presolved problem has: 5 rows 6 cols 12 non-zeros

<table>
<thead>
<tr>
<th>Its</th>
<th>Obj Value</th>
<th>S</th>
<th>Ninf</th>
<th>Nneg</th>
<th>Sum Inf</th>
<th>Time</th>
</tr>
</thead>
</table>
Uncrunching matrix

Optimal solution found

optimal LP solution found: objective value 153.675

--- Spawning solver : CPLEX

GAMS/Cplex Jan 19, 2004 LNX.CP.CP 21.3 025.027.041.LXI For Cplex 9.0
Cplex 9.0.0, GAMS Link 25

Reading data...
Starting Cplex...
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.00 sec.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Dual Objective</th>
<th>In Variable</th>
<th>Out Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73.125000</td>
<td>x(seattle.new-york)</td>
<td>demand(new-york) slack</td>
</tr>
<tr>
<td>2</td>
<td>119.025000</td>
<td>x(seattle.chicago)</td>
<td>demand(chicago) slack</td>
</tr>
<tr>
<td>3</td>
<td>153.675000</td>
<td>x(san-diego.topeka)</td>
<td>demand(topeka) slack</td>
</tr>
<tr>
<td>4</td>
<td>153.675000</td>
<td>x(san-diego.new-york)</td>
<td>supply(seattle) slack</td>
</tr>
</tbody>
</table>

Optimal solution found.
Objective : 153.675000

--- BENCH SUMMARY:

<table>
<thead>
<tr>
<th>Solver</th>
<th>Modstat</th>
<th>Solstat</th>
<th>Objective</th>
<th>ResUsd</th>
<th>Examiner</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDMLP</td>
<td>1</td>
<td>1</td>
<td>153.675</td>
<td>0.000</td>
<td>n/a</td>
</tr>
<tr>
<td>MINOS</td>
<td>1</td>
<td>1</td>
<td>153.675</td>
<td>0.000</td>
<td>n/a</td>
</tr>
<tr>
<td>XPRESS</td>
<td>1</td>
<td>1</td>
<td>153.675</td>
<td>0.040</td>
<td>n/a</td>
</tr>
<tr>
<td>CPLEX</td>
<td>1</td>
<td>1</td>
<td>153.675</td>
<td>0.000</td>
<td>n/a</td>
</tr>
</tbody>
</table>

--- Restarting execution
--- trnsport.gms(66) 0 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion

### 5.2 The BENCH Listing File

The BENCH listing file is similar to the standard GAMS format. It contains a benchmark summary of all solvers called. The regular solve summary for BENCH does not return a solution (although the solution of the final solve can be returned using the `returnlastsol` option). For the example below we use the batchdea model from the GAMS model library using the solvers SBB and DICOPT. We specify the two solvers using a `bench.opt` option file with the entry:

```plaintext
solvers sbb.1 dicopt
```

Note that SBB will use a solver option file called `sbb.opt`. 
SOLVE SUMMARY

MODEL batch OBJECTIVE cost
TYPE MINLP DIRECTION MINIMIZE
SOLVER BENCH FROM LINE 183

**** SOLVER STATUS 1 NORMAL COMPLETION
**** MODEL STATUS 14 NO SOLUTION RETURNED
**** OBJECTIVE VALUE 0.0000

RESOURCE USAGE, LIMIT 0.000 1000.000
ITERATION COUNT, LIMIT 0 10000
Reading user supplied options file /home/models/bench.opt
Processing...
> solvers sbb.1 dicopt

Note that the model status return code for BENCH itself is always SOLVER STATUS 1 and MODEL STATUS 14, since BENCH itself does not return a solution by default. To obtain the status codes and solution information of the last solver, the returnlastsol option can be enabled.

In addition the listing file contains complete solve summary information for each solver called. Also, note that the option file used for SBB and its contents are echoed to the SBB summary.

BENCHMARK SUMMARY

SOLVER SBB
SOLVER STATUS 1 NORMAL COMPLETION
MODEL STATUS 8 INTEGER SOLUTION
OBJECTIVE VALUE 167427.6571
RESOURCE USAGE, LIMIT 0.080 1000.000
ITERATION COUNT, LIMIT 139 100000
EVALUATION ERRORS, LIMIT 0 0
OPTION FILE sbb.opt
Reading user supplied options file sbb.opt
Processing...
> rootsolver conopt2
> subsolver snopt

SOLVER DICOPT
SOLVER STATUS 1 NORMAL COMPLETION
MODEL STATUS 8 INTEGER SOLUTION
OBJECTIVE VALUE 167427.6571
RESOURCE USAGE, LIMIT 0.100 999.920
ITERATION COUNT, LIMIT 117 99861
EVALUATION ERRORS, LIMIT 0 0

Note that the listing file does not contain detailed solution information since BENCH does not return any values.

6 Interrupting BENCH with Ctrl-C

BENCH passes all Control-C (Ctrl-C) signals to the respective subsolvers. If a terminate signal via Ctrl-C is sent in the middle of a solver run (i.e. not initially when the solver begins execution), the individual subsolver is terminated.
To terminate not only the subsolver but also BENCH, a Ctrl-C signal should be sent at the beginning of a solver’s execution. Thus, several Ctrl-C in rapid succession will terminate BENCH.

Benchmark summary information will be written to the listing file for each solver that has successfully completed without any signal interrupt.

7 Benchmark Example

In this section we will give a small example showing how to use the BENCH solver and automate the subsequent analysis using the PAVER Server. In particular, we will run the three versions of CONOPT (CONOPT1, CONOPT2, and CONOPT3, as well as CONOPT3 with no scaling) on the default instance of the COPS models for nonlinear programming. We will use the 17 models available from the GAMS Model Library.

First we need to extract all of the models from the GAMS Model Library. We can create a file which will extract these automatically. Create a file called getcops.gms with the entries below:

```
$call gamslib camshape
$call gamslib catmix
$call gamslib chain
$call gamslib elec
$call gamslib flowchan
$call gamslib gasoil
$call gamslib glider
$call gamslib jbearing
$call gamslib lnts
$call gamslib methanol
$call gamslib minsurf
$call gamslib pinene
$call gamslib polygon
$call gamslib popdynm
$call gamslib robot
$call gamslib rocket
$call gamslib torsion
```

Running the file using gams getcops.gms extracts the models. Then create a BENCH solver option file called bench.opt with the entries

```
solvers conopt1 conopt2 conopt3 conopt3.1
paver 1
```

The first entry tells BENCH to run the solvers CONOPT1, CONOPT2, and CONOPT3 and then CONOPT3 with the option file conopt3.opt. The second entry tells BENCH to create PAVER trace files. These can be submitted to the PAVER server for automated performance analysis. Now create an option file conopt3.opt with the entry

```
1sscal f
```

which tells CONOPT3 not to use scaling.

We can now run the models in batch mode, for example by creating a GAMS batch file runcops.gms with the following entries:

```
$call gams camshape.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams catmix.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams chain.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams elec.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams flowchan.gms nlp=bench optfile=1 reslim=10 domlim=99999
```
Running the file using the command gams runcops.gms runs all models with all three solvers through the GAMS/BENCH solver. Furthermore, three PAVER trace files are created: conopt1.pvr, conopt2.pvr, conopt3.pvr and conopt3-1.pvr, where the latter is for CONOPT3 with no scaling. Users can then submit the three trace files to the PAVER Server for automated analysis.

The resulting performance plot in the following figure shows the efficiency of each solver/solver option.

![Performance Profile](image)

Figure 27.1: PAVER: Process Overview
COIN-OR BONMIN (Basic Open-source Nonlinear Mixed Integer programming) is an open-source solver for mixed-integer nonlinear programming (MINLPs), implementing branch-and-bound, branch-and-cut, and outer approximation algorithms. The code has been developed as part of a collaboration between Carnegie Mellon University and IBM Research. The COIN-OR project leader for BONMIN is Pierre Bonami.

BONMIN can handle mixed-integer nonlinear programming models which functions should be twice continuously differentiable. The BONMIN link in GAMS supports continuous, binary, and integer variables, special ordered sets, branching priorities, but no semi-continuous or semi-integer variables (see chapter 17.1 of the GAMS User’s Guide).

BONMIN implements six different algorithms for solving MINLPs:

- **B-BB (default):** a simple branch-and-bound algorithm based on solving a continuous nonlinear program at each node of the search tree and branching on integer variables [11]; this algorithm is similar to the one implemented in the solver SBB

- **B-OA:** an outer-approximation based decomposition algorithm based on iterating solving and improving of a MIP relaxation and solving NLP subproblems [9, 10]; this algorithm is similar to the one implemented in the solver DICOPT

- **B-QG:** an outer-approximation based branch-and-cut algorithm based on solving a continuous linear program at each node of the search tree, improving the linear program by outer approximation, and branching on integer variables [12].
• B-Hyb: a branch-and-bound algorithm which is a hybrid of B-BB and B-QG and is based on solving either a continuous nonlinear or a continuous linear program at each node of the search tree, improving the linear program by outer approximation, and branching on integer variables [6]

• B-ECP: a Kelley’s outer-approximation based branch-and-cut algorithm inspired by the settings used in the solver FilMINT [1]

• B-iFP: an iterated feasibility pump algorithm [7] The algorithms are exact when the problem is convex, otherwise they are heuristics.

For convex MINLPS, experiments on a reasonably large test set of problems have shown that B-Hyb is the algorithm of choice (it solved most of the problems in 3 hours of computing time). Nevertheless, there are cases where B-OA (especially when used with CPLEX as MIP subproblem solver) is much faster than B-Hyb and others where B-BB is interesting. B-QG and B-ECP corresponds mainly to a specific parameter setting of B-Hyb but they can be faster in some cases. B-iFP is more tailored at finding quickly good solutions to very hard convex MINLP. For nonconvex MINLPS, it is strongly recommended to use B-BB (the outer-approximation algorithms have not been tailored to treat nonconvex problems at this point). Although even B-BB is only a heuristic for such problems, several options are available to try and improve the quality of the solutions it provides (see below).

NLPs are solved in BONMIN by IPOPT, which can use MUMPS [2, 3] (currently the default) or MKL PARDISO [13, 14] (only Linux and Windows) as linear solver. In the commercially licensed GAMS/BONMINH version, also the linear solvers MA27, MA57, HSL_MA86, and HSL_MA97 from the Harwell Subroutines Library (HSL) are available in IPOPT. In this case, the default linear solver in IPOPT is MA27.

For more information on BONMIN we refer to [4, 6–8] and the BONMIN web site. Most of the BONMIN documentation in this section is taken from the BONMIN manual [5].

1 Usage

The following statement can be used inside your GAMS program to specify using BONMIN:

```
Option MINLP = BONMIN; { or Option MIQCP = BONMIN; }
```

This statement should appear before the `Solve` statement. If BONMIN was specified as the default solver during GAMS installation, the above statement is not necessary.

To use BONMINH, one should use the statement

```
Option MINLP = BONMINH; { or Option MIQCP = BONMINH; }
```

GAMS/BONMIN currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/BONMIN with BCH, please consider to use a GAMS system of version ≤ 23.3.

1.1 Specification of Options

A BONMIN options file contains both IPOPT and BONMIN options, for clarity all BONMIN options should be preceded with the prefix bonmin. The scheme to name option files is the same as for all other GAMS solvers. The format of the option file is the same as for IPOPT.

The most important option in BONMIN is the choice of the solution algorithm. This can be set by using the option named algorithm which can be set to B-BB, B-OA, B-QG, B-Hyb, B-ECP, or B-iFP (its default value is B-BB). Depending on the value of this option, certain other options may be available or not, cf. Section List of all BONMIN Options.

An example of a bonmin.opt file is the following:

```
bonmin.algorithm B-Hyb
bonmin.oa_log_level 4
print_level 6
```
This sets the algorithm to be used to the hybrid algorithm, the level of outer approximation related output to 4, and sets the print level for IPOPT to 6.

GAMS/BONMIN understands currently the following GAMS parameters: \texttt{reslim} (time limit), \texttt{iterlim} (iteration limit), \texttt{nodlim} (node limit), \texttt{cutoff}, \texttt{optca} (absolute gap tolerance), and \texttt{optcr} (relative gap tolerance).

Further, the option \texttt{threads} can be used to control the number of threads used in the linear algebra routines of IPOPT, see \textit{The linear solver in IPOPT} in the IPOPT manual for details.

### 1.2 Passing options to local search based heuristics and OA generators

Several parts of the algorithms in BONMIN are based on solving a simplified version of the problem with another instance of BONMIN: Outer Approximation Decomposition (called in B-Hyb at the root node) and Feasibility Pump for MINLP (called in B-Hyb or B-BB at the root node), RINS, RENS, Local Branching.

In all these cases, one can pass options to the sub-algorithm used through the option file. The basic principle is that the \texttt{bonmin} prefix is replaced with a prefix that identifies the sub-algorithm used:

- \texttt{oa\_decomposition}. to pass options to Outer Approximation Decomposition,
- \texttt{pump\_for\_minlp}. to pass options to Feasibility Pump for MINLP,
- \texttt{rins}. to pass options to RINS,
- \texttt{rens}. to pass options to RENS,
- \texttt{local\_branch}. to pass options to Local Branching.

For example, to run a maximum of 60 seconds of feasibility pump (FP) for MINLP until 6 solutions are found at the beginning of the hybrid algorithm, one sets the following options:

```
bonmin.algorithm B-Hyb
bonmin.pump_for_minlp yes  # tells to run FP for MINLP
pump_for_minlp.time_limit 60  # set a time limit for the pump
pump_for_minlp.solution_limit 6  # set a solution limit
```

Note that the actual solution and time limit will be the minimum of the global limits set for BONMIN.

A slightly more complicated set of options may be used when using RINS. Say for example that one wants to run RINS inside B-BB. Each time RINS is called one wants to solve the small-size MINLP generated using B-QG (one may run any algorithm available in BONMIN for solving an MINLP) and wants to stop as soon as B-QG found one solution. To achieve this, one sets the following options

```
bonmin.algorithm B-BB
bonmin.heuristic_rins yes
rins.algorithm B-QG
rins.solution_limit 1
```

This example shows that it is possible to set any option used in the sub-algorithm to be different than the one used for the main algorithm.

In the context of outer-approximation (OA) and feasibility pump for MINLP, a standard MILP solver is used. Several options are available for configuring this MILP solver. BONMIN allows a choice of different MILP solvers through the option \texttt{milp\_solver}. Values for this option are: \texttt{Cbc\_D}, which uses CBC with its default settings, \texttt{Cbc\_Par}, which uses a version of CBC that can be parameterized by the user, and \texttt{Cplex}, which uses CPLEX with its default settings. The options that can be set in \texttt{Cbc\_Par} are the number of strong-branching candidates, the number of branches before pseudo costs are to be trusted, and the frequency of the various cut generators, cf. Section \textit{List of all BONMIN Options} for details. To use the \texttt{Cplex} option, a valid CPLEX licence (standalone or GAMS/CPLEX) is required.
1.3 Getting good solutions to nonconvex problems

To solve a problem with nonconvex constraints, one should only use the branch-and-bound algorithm B-BB.

A few options have been designed in BONMIN specifically to treat problems that do not have a convex continuous relaxation. In such problems, the solutions obtained from IPOPT are not necessarily globally optimal, but are only locally optimal. Also the outer-approximation constraints are not necessarily valid inequalities for the problem. No specific heuristic method for treating nonconvex problems is implemented yet within the OA framework. But for the pure branch-and-bound B-BB, a few options have been implemented while having in mind that lower bounds provided by IPOPT should not be trusted and with the goal of trying to get good solutions. Such options are at a very experimental stage.

First, in the context of nonconvex problems, IPOPT may find different local optima when started from different starting points. The two options num resolve at root and num resolve at node allow for solving the root node or each node of the tree, respectively, with a user-specified number of different randomly-chosen starting points, saving the best solution found. Note that the function to generate a random starting point is very naive: it chooses a random point (uniformly) between the bounds provided for the variable. In particular if there are some functions that can not be evaluated at some points of the domain, it may pick such points, and so it is not robust in that respect.

Secondly, since the solution given by IPOPT does not truly give a lower bound, the fathoming rule can be changed to continue branching even if the solution value to the current node is worse than the best-known solution. This is achieved by setting allowable gap, allowable fraction gap, and cutoff decr to negative values.

1.4 IPOPT options changed by BONMIN

IPOPT has a very large number of options, see Section List of IPOPT Options to get a complete description. To use IPOPT more efficiently in the context of MINLP, BONMIN changes some IPOPT options from their default values, which may help to improve IPOPT’s warm-starting capabilities and its ability to prove quickly that a subproblem is infeasible. These are settings that IPOPT does not use for ordinary NLP problems. Note that options set by the user in an option file will override these settings.

- **mu strategy** and **mu oracle** are set, respectively, to adaptive and probing by default. These are strategies in IPOPT for updating the barrier parameter. They were found to be more efficient in the context of MINLP.

- **gamma phi** and **gamma theta** are set to 1E-8 and 1E-4, respectively. This has the effect of reducing the size of the filter in the line search performed by IPOPT.

- **required infeasibility reduction** is set to 0.1. This increases the required infeasibility reduction when IPOPT enters the restoration phase and should thus help to detect infeasible problems faster.

- **expect infeasible problem** is set to yes, which enables some heuristics to detect infeasible problems faster.

- **warm start init point** is set to yes when a full primal/dual starting point is available (generally for all the optimizations after the continuous relaxation has been solved).

- **print level** is set to 0 by default to turn off IPOPT output (except for the root node, which print level is controlled by the BONMIN option BONMINnlp log at root nlp log at root).

- **bound relax factor** is set to 1E-10. All of the bounds of the problem are relaxed by this factor. This may cause some trouble when constraint functions can only be evaluated within their bounds. In such cases, this option should be set to 0.

2 List of all BONMIN Options

The following tables give lists of options together with their types, default values, and availability in each of the main algorithms. The column labeled Cbc Par indicates the options that can be used to parametrize the MILP subsolver in the context of OA and FP.
2.1 Algorithm choice

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc_Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>string</td>
<td>B-BB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

2.2 Branch-and-bound options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc_Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowable_fraction_gap</td>
<td>Q</td>
<td>GAMS optcr</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>allowable_gap</td>
<td>Q</td>
<td>GAMS optca</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cutoff</td>
<td>Q</td>
<td>GAMS cutoff</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cutoff_decr</td>
<td>Q</td>
<td>10^{-5}</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>enable_dynamic_nlp</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>integer_tolerance</td>
<td>Q</td>
<td>10^{-6}</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>iteration_limit</td>
<td>Z</td>
<td>GAMS iterlim</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nlp_failure_behavior</td>
<td>string</td>
<td>stop</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>node_comparison</td>
<td>string</td>
<td>best-bound</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>node_limit</td>
<td>Z</td>
<td>GAMS nodlim</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>num_cut_passes</td>
<td>Z</td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>num_cut_passes_at_root</td>
<td>Z</td>
<td>20</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>number_before_trust</td>
<td>Z</td>
<td>8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>number_strong_branch</td>
<td>Z</td>
<td>20</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>random_generator_seed</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>read_solution_file</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>solution_limit</td>
<td>Z</td>
<td>∞</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>time_limit</td>
<td>Q</td>
<td>GAMS reslim</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>tree_search_strategy</td>
<td>string</td>
<td>probed-dive</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>variable_selection</td>
<td>string</td>
<td>strong-branching</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>

2.3 ECP cuts generation

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc_Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecp_abs_tol</td>
<td>Q</td>
<td>10^{-6}</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ecp_max_rounds</td>
<td>Z</td>
<td>5</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ecp_probability_factor</td>
<td>Q</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ecp_rel_tol</td>
<td>Q</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
### 2.4 Feasibility checker using OA cuts

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>feas_check_cut_types</td>
<td>string</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>feas_check_discard_policy</td>
<td>string</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>generate_benders_after_so_many_oa</td>
<td>Z</td>
<td>5000</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2.5 MILP Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpx_parallel_strategy</td>
<td>Z</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>milp_solver</td>
<td>string</td>
<td>Cbc_D</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>milp_strategy</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>number_cpx_threads</td>
<td>Z</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

### 2.6 MILP cutting planes in hybrid algorithm

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>2mir_cuts</td>
<td>Z</td>
<td>0</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Gomory_cuts</td>
<td>Z</td>
<td>−5</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>clique_cuts</td>
<td>Z</td>
<td>−5</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cover_cuts</td>
<td>Z</td>
<td>0</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>flow_cover_cuts</td>
<td>Z</td>
<td>−5</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lift_and_project_cuts</td>
<td>Z</td>
<td>0</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mir_cuts</td>
<td>Z</td>
<td>−5</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>reduce_and_split_cuts</td>
<td>Z</td>
<td>0</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 2.7 NLP interface

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>solvefinal</td>
<td>string</td>
<td>yes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>warm_start</td>
<td>string</td>
<td>none</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2.8 NLP solution robustness
### 2.9 NLP solves in hybrid algorithm (B-Hyb)

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlp_solve_frequency</td>
<td>Z</td>
<td>10</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>nlp_solve_max_depth</td>
<td>Z</td>
<td>10</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>nlp_solves_per_depth</td>
<td>Q</td>
<td>10^100</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### 2.10 Nonconvex problems

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>coeff_var_threshold</td>
<td>Q</td>
<td>0.1</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>dynamic_def_cutoff_decr</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>first_perc_for_cutoff_decr</td>
<td>Q</td>
<td>−0.02</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>max_consecutive_infeasible</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>num_resolve_at_infeasibles</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>num_resolve_at_node</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>num_resolve_at_root</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>second_perc_for_cutoff_decr</td>
<td>Q</td>
<td>−0.05</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### 2.11 Outer Approximation Decomposition (B-OA)

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>oa_decomposition</td>
<td>string</td>
<td>no</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### 2.12 Outer Approximation cuts generation

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc.Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_only_violated_oa</td>
<td>string</td>
<td>no</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>oa_cuts_scope</td>
<td>string</td>
<td>global</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>oa_rhs_relax</td>
<td>Q</td>
<td>10^-8</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>tiny_element</td>
<td>Q</td>
<td>10^-8</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 2.13 Output and Loglevel

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc_Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb_log_interval</td>
<td>Z</td>
<td>100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>bb_log_level</td>
<td>Z</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>fp_log_frequency</td>
<td>Q</td>
<td>100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>fp_log_level</td>
<td>Z</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lp_log_level</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>milp_log_level</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nlp_at_root</td>
<td>Z</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nlp_log_level</td>
<td>Z</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>oa_cuts_log_level</td>
<td>Z</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>oa_log_frequency</td>
<td>Q</td>
<td>100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>oa_log_level</td>
<td>Z</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>print_funceval_statistics</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>solvtrace</td>
<td>string</td>
<td>...</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>solvtracenodefreq</td>
<td>Z</td>
<td>100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>solvtracestimefreq</td>
<td>Q</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 2.14 Primal Heuristics

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc_Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>feasibility_pump_objective_norm</td>
<td>Z</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>fp_pass_infeasible</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>heuristic_RINS</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>heuristic_dive_MIP_fractional</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>heuristic_dive_MIP_vectorLength</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>heuristic_dive_fractional</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>heuristic_dive_vectorLength</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>heuristic_feasibility_pump</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>pump_for_minlp</td>
<td>string</td>
<td>no</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 2.15 Strong branching setup

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>B-BB</th>
<th>B-OA</th>
<th>B-QG</th>
<th>B-Hyb</th>
<th>B-Ecp</th>
<th>B-iFP</th>
<th>Cbc_Par</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidate_sort_criterion</td>
<td>string</td>
<td>best-as-cost</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>maxmin_crit_have_sol</td>
<td>Q</td>
<td>0.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>maxmin_crit_no_sol</td>
<td>Q</td>
<td>0.7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
3 Detailed Options Description

In the following we give a detailed description of all BONMIN options.

2mir_cuts (integer): Frequency (in terms of nodes) for generating 2-MIR cuts in branch-and-cut

If $k > 0$, cuts are generated every $k$ nodes, if $-99 < k < 0$ cuts are generated every $-k$ nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if $k = -99$ generate cuts only at the root node, if $k = 0$ or 100 do not generate cuts.

(default = 0)

acceptable_compl_inf_tol (real): 'Acceptance' threshold for the complementarity conditions.

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also acceptable_tol.

(default = 0.01)

acceptable_constr_viol_tol (real): 'Acceptance' threshold for the constraint violation.

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also acceptable_tol.

(default = 0.01)

acceptable_dual_inf_tol (real): 'Acceptance' threshold for the dual infeasibility.

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also acceptable_tol.

(default = 1e+10)

acceptable_iter (integer): Number of 'acceptable' iterates before triggering termination.

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled.

(default = 15)

acceptable_obj_change_tol (real): 'Acceptance' stopping criterion based on objective function change.

If the relative change of the objective function (scaled by Max(1,|f(x)|)) is less than this value, this part of the acceptable tolerance termination is satisfied; see also acceptable_tol. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility.

(default = 1e+20)

acceptable_tol (real): 'Acceptable' convergence tolerance (relative).

Determines which (scaled) overall optimality error is considered to be "acceptable." There are two levels of termination criteria. If the usual "desired" tolerances (see tol, dual_inf_tol etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the
"desired" level of accuracy.
(default = 1e-06)

**accept_after_max_steps (integer):** Accept a trial point after maximal this number of steps.
Even if it does not satisfy line search conditions.
(default = -1)

**accept_every_trial_step (string):** Always accept the first trial step.
Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees.
(default = no)

- no don’t arbitrarily accept the full step
- yes always accept the full step

**adaptive_mu_globalization (string):** Globalization strategy for the adaptive mu selection mode.
To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive".)
(default = obj-constr-filter)

- kkt-error nonmonotone decrease of kkt-error
- never-monotone-mode disables globalization
- obj-constr-filter 2-dim filter for objective and constraint violation

**adaptive_mu_kkterror_red_fact (real):** Sufficient decrease factor for 'kkt-error' globalization strategy.
For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease.
Range: [0, 1]
(default = 0.9999)

**adaptive_mu_kkterror_red_iters (integer):** Maximum number of iterations requiring sufficient progress.
For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.
(default = 4)

**adaptive_mu_kkt_norm_type (string):** Norm used for the KKT error in the adaptive mu globalization strategies.
When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this options is also used in the QualityFunctionMuOracle.
(default = 2-norm-squared)

- 1-norm use the 1-norm (abs sum)
- 2-norm use 2-norm
- 2-norm-squared use the 2-norm squared (sum of squares)
- max-norm use the infinity norm (max)

**adaptive_mu_monotone_init_factor (real):** Determines the initial value of the barrier parameter when switching to the monotone mode.
When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".
(default = 0.8)
**adaptive_mu_restore_previous_iterate** (*string*): Indicates if the previous iterate should be restored if the monotone mode is entered.

    When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).
    (default = no)
    no don’t restore accepted iterate
    yes restore accepted iterate

**add_only_violated_oa** (*string*): Do we add all OA cuts or only the ones violated by current point?

    (default = no)
    no Add all cuts
    yes Add only violated cuts

**algorithm** (*string*): Choice of the algorithm.

    This will preset some of the options of bonmin depending on the algorithm choice.
    (default = B-BB)
    b-bb simple branch-and-bound algorithm,
    b-ecp ECP cuts based branch-and-cut a la FilMINT.
    b-hyb hybrid outer approximation based branch-and-cut,
    b-ifp Iterated Feasibility Pump for MINLP.
    b-oa OA Decomposition algorithm,
    b-qg Quesada and Grossmann branch-and-cut algorithm,

**allowable_fraction_gap** (*real*): Specify the value of relative gap under which the algorithm stops.

    Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of the absolute value of the best known solution value.
    (default = 0.1)

**allowable_gap** (*real*): Specify the value of absolute gap under which the algorithm stops.

    Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this.
    (default = 0)

**alpha_for_y** (*string*): Method to determine the step size for constraint multipliers.

    This option determines how the step size (alpha_y) will be calculated when updating the constraint multipliers.
    (default = primal)
    acceptor Call LSAcceptor to get step size for y
    bound-mult use step size for the bound multipliers (good for LPs)
    dual-and-full use the dual step size, and full step if delta_x ≤ alpha_for_y_tol
    full take a full step of size one
    max use the max of primal and bound multipliers
    min use the min of primal and bound multipliers
    min-dual-infeas choose step size minimizing new dual infeasibility
    primal use primal step size
    primal-and-full use the primal step size, and full step if delta_x ≤ alpha_for_y_tol
    safer-min-dual-infeas like ’min_dual_infeas’, but safeguarded by ’min’ and ’max’
alpha_for_y_tol (real): Tolerance for switching to full equality multiplier steps.

This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

(default = 10)

alpha_min_frac (real): Safety factor for the minimal step size (before switching to restoration phase).

(This is gamma_alpha in Eqn. (20) in the implementation paper.)

Range: [0, 1]

(default = 0.05)

alpha_red_factor (real): Fractional reduction of the trial step size in the backtracking line search.

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: [0, 1]

(default = 0.5)

barrier_tol_factor (real): Factor for mu in barrier stop test.

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. (This is kappa_epsilon in implementation paper).

(default = 10)

bb_log_interval (integer): Interval at which node level output is printed.

Set the interval (in terms of number of nodes) at which a log on node resolutions (consisting of lower and upper bounds) is given.

(default = 100)

bb_log_level (integer): specify main branch-and-bound log level.

Set the level of output of the branch-and-bound : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high

Range: [0, 5]

(default = 1)

bound_frac (real): Desired minimum relative distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

(default = 0.01)

bound_mult_init_method (string): Initialization method for bound multipliers

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound_mult_init_val". If "mu-based" is chosen, the each value is initialized to the the value of "mu_init" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.

(default = constant)

constant set all bound multipliers to the value of bound_mult_init_val

mu-based initialize to mu_init/x_slack

bound_mult_init_val (real): Initial value for the bound multipliers.

All dual variables corresponding to bound constraints are initialized to this value.

(default = 1)

bound_mult_reset_threshold (real): Threshold for resetting bound multipliers after the restoration phase.
After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

(default = 1000)

**bound_push** *(real):* Desired minimum absolute distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

(default = 0.01)

**bound_relax_factor** *(real):* Factor for initial relaxation of the bounds.

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then then bounds relaxation is disabled. (See Eqn.(35) in implementation paper.)

(default = 1e-10)

**candidate_sort_criterion** *(string):* Choice of the criterion to choose candidates in strong-branching

(default = best-ps-cost)

  - best-ps-cost Sort by decreasing pseudo-cost
  - least-fractional Sort by increasing integer infeasibility
  - most-fractional Sort by decreasing integer infeasibility
  - worst-ps-cost Sort by increasing pseudo-cost

**check_derivatives_for_naninf** *(string):* Indicates whether it is desired to check for Nan/Inf in derivative matrices

Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an invalid number is detected, the matrix is written to output with print_level corresponding to J_MORE_DETAILED; so beware of large output!

(default = no)

  - no Don’t check (faster).
  - yes Check Jacobians and Hessian for Nan and Inf.

**clique_cuts** *(integer):* Frequency (in terms of nodes) for generating clique cuts in branch-and-cut

See option 2mir_cuts for a detailed description.

(default = -5)

**coeff_var_threshold** *(real):* Coefficient of variation threshold (for dynamic definition of cutoff_decr).

(default = 0.1)

**compl_inf_tol** *(real):* Desired threshold for the complementarity conditions.

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.

(default = 0.0001)

**constraintViolation_norm_type** *(string):* Norm to be used for the constraint violation in the line search.

Determines which norm should be used when the algorithm computes the constraint violation in the line search.

(default = 1-norm)

  - 1-norm use the 1-norm
  - 2-norm use the 2-norm
  - max-norm use the infinity norm
**constr_mult_init_max (real):** Maximum allowed least-square guess of constraint multipliers.

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This option is also used when initializing the restoration phase. By default, “resto.constr_mult_init_max” (the one used in RestoIterateInitializer) is set to zero.

(default = 1000)

**constr_mult_reset_threshold (real):** Threshold for resetting equality and inequality multipliers after restoration phase.

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.

(default = 0)

**constr_viol_tol (real):** Desired threshold for the constraint violation.

Absolute tolerance on the constraint violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold.

(default = 0.0001)

**corrector_compl_avrg_red_fact (real):** Complementarity tolerance factor for accepting corrector step (unsupported!).

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted.

(default = 1)

**corrector_type (string):** The type of corrector steps that should be taken (unsupported!).

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried.

(default = none)
  
  affine corrector step towards mu=0
  none no corrector
  primal-dual corrector step towards current mu

**cover_cuts (integer):** Frequency (in terms of nodes) for generating cover cuts in branch-and-cut

See option 2mir_cuts for a detailed description.

(default = 0)

**cpx_parallel_strategy (integer):** Strategy of parallel search mode in CPLEX.

-1 = opportunistic, 0 = automatic, 1 = deterministic (refer to CPLEX documentation)

Range: [-1, 1]

(default = 0)

**cutoff (real):** Specify cutoff value.

cutoff should be the value of a feasible solution known by the user (if any). The algorithm will only look for solutions better than cutoff.

Range: [-1e+100, 1e+100]

(default = 1e+100)

**cutoff_decr (real):** Specify cutoff decrement.

Specify the amount by which cutoff is decremented below a new best upper-bound (usually a small positive value but in non-convex problems it may be a negative value).

Range: [-1e+10, 1e+10]

(default = 1e^-05)

**delta (real):** Multiplier for constraint violation in the switching rule.
(See Eqn. (19) in the implementation paper.)
(default = 1)

**dependency_detection_with_rhs (string):** Indicates if the right hand sides of the constraints should be considered during dependency detection

(default = no)

- no: only look at gradients
- yes: also consider right hand side

**dependency_detector (string):** Indicates which linear solver should be used to detect linearly dependent equality constraints.

The default and available choices depend on how Ipopt has been compiled. This is experimental and does not work well.

(default = none)

- ma28: use MA28
- mumps: use MUMPS
- none: don’t check; no extra work at beginning

**diverging_iterates_tol (real):** Threshold for maximal value of primal iterates.

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.

(default = 1e+20)

**dual_inf_tol (real):** Desired threshold for the dual infeasibility.

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.

(default = 1)

**dynamic_def_cutoff_decr (string):** Do you want to define the parameter cutoff_decr dynamically?

(default = no)

- no
- yes

**ecp_abs_tol (real):** Set the absolute termination tolerance for ECP rounds.

(default = 1e-06)

**ecp_max_rounds (integer):** Set the maximal number of rounds of ECP cuts.

(default = 5)

**ecp_probability_factor (real):** Factor appearing in formula for skipping ECP cuts.

Choosing -1 disables the skipping.

(default = 10)

**ecp_rel_tol (real):** Set the relative termination tolerance for ECP rounds.

(default = 0)

**enable_dynamic_nlp (string):** Enable dynamic linear and quadratic rows addition in nlp

(default = no)

- no
- yes

**eta_phi (real):** Relaxation factor in the Armijo condition.

(See Eqn. (20) in the implementation paper)
evaluate_orig_obj_at_resto_trial (string): Determines if the original objective function should be evaluated at restoration phase trial points.

Setting this option to "yes" makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.

(expect_infeasible_problem (string): Enable heuristics to quickly detect an infeasible problem.

This options is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.

(expect_infeasible_problem_ctol (real): Threshold for disabling 'expect_infeasible_problem' option.

If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this options is set to 0.

(expect_infeasible_problem_ytol (real): Multiplier threshold for activating 'expect_infeasible_problem' option.

If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.

(fast_step_computation (string): Indicates if the linear system should be solved quickly.

If set to yes, the algorithm assumes that the linear system that is solved to obtain the search direction, is solved sufficiently well. In that case, no residuals are computed, and the computation of the search direction is a little faster.

(feasibility_pump_objective_norm (integer): Norm of feasibility pump objective function

Range: [1, 2]

(feas_check_cut_types (string): Choose the type of cuts generated when an integer feasible solution is found

If it seems too much memory is used should try Benders to use less

(benders Generate a single Benders cut.
outer-approx Generate a set of Outer Approximations cuts.

**feas_check_discard_policy (string):** How cuts from feasibility checker are discarded

Normally to avoid cycle cuts from feasibility checker should not be discarded in the node where they are generated. However Cbc sometimes does it if no care is taken which can lead to an infinite loop in Bonmin (usually on simple problems). To avoid this one can instruct Cbc to never discard a cut but if we do that for all cuts it can lead to memory problems. The default policy here is to detect cycles and only then impose to Cbc to keep the cut. The two other alternative are to instruct Cbc to keep all cuts or to just ignore the problem and hope for the best

(default = detect-cycles)

- detect-cycles Detect if a cycle occurs and only in this case force not to discard.
- keep-all Force cuts from feasibility checker not to be discarded (memory hungry but sometimes better).
- treated-as-normal Cuts from memory checker can be discarded as any other cuts (code may cycle then)

**filmint_ecp_cuts (integer):** Specify the frequency (in terms of nodes) at which some a la filmint ecp cuts are generated.

A frequency of 0 amounts to to never solve the NLP relaxation.

(default = 0)

**filter_margin_fact (real):** Factor determining width of margin for obj-constr-filter adaptive globalization strategy.

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: (new obj) < (filter obj) - filter_margin_fact*(new constr-viol) OR (new constr-viol) < (filter constr-viol) - filter_margin_fact*(new constr-viol). For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: [0, 1]

(default = 1e-05)

**filter_max_margin (real):** Maximum width of margin in obj-constr-filter adaptive globalization strategy.

(default = 1)

**filter_reset_trigger (integer):** Number of iterations that trigger the filter reset.

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

(default = 5)

**first_hessian_perturbation (real):** Size of first x-s perturbation tried.

The first value tried for the x-s perturbation in the inertia correction scheme.(This is delta_0 in the implementation paper.)

(default = 0.0001)

**first_perc_for_cutoff_decr (real):** The percentage used when, the coeff of variance is smaller than the threshold, to compute the cutoff_decr dynamically.

(default = ~0.02)

**fixed_mu_oracle (string):** Oracle for the barrier parameter when switching to fixed mode.

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy").

(default = average_compl)

- average_compl base on current average complementarity
- loqo LOQO’s centrality rule
- probing Mehrotra’s probing heuristic
quality-function minimize a quality function

**fixed_variable_treatment (string):** Determines how fixed variables should be handled.

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to "bound_relax_factor"). For both "make_constraints" and "relax_bounds", bound multipliers are computed for the fixed variables.

(default = make_parameter)

- make_constraint Add equality constraints fixing variables
- make_parameter Remove fixed variable from optimization variables
- relax_bounds Relax fixing bound constraints

**flow_cover_cuts (integer):** Frequency (in terms of nodes) for generating flow cover cuts in branch-and-cut

See option 2mir_cuts for a detailed description.

(default = -5)

**fp_log_frequency (real):** display an update on lower and upper bounds in FP every n seconds

(default = 100)

**fp_log_level (integer):** specify FP iterations log level.

Set the level of output of OA decomposition solver: 0 - none, 1 - normal, 2 - verbose

Range: [0, 2]

(default = 1)

**fp_pass_infeasible (string):** Say whether feasibility pump should claim to converge or not

(default = no)

- no When master MILP is infeasible just bail out (don’t stop all algorithm). This is the option for using in B-Hyb.
- yes Claim convergence, numerically dangerous.

**gamma_phi (real):** Relaxation factor in the filter margin for the barrier function.

(See Eqn. (18a) in the implementation paper.)

Range: [0, 1]

(default = 1e-08)

**gamma_theta (real):** Relaxation factor in the filter margin for the constraint violation.

(See Eqn. (18b) in the implementation paper.)

Range: [0, 1]

(default = 0.0001)

**generate_benders_after_so_many_oa (integer):** Specify that after so many oa cuts have been generated Benders cuts should be generated instead.

It seems that sometimes generating too many oa cuts slows down the optimization compared to Benders due to the size of the LP. With this option we specify that after so many OA cuts have been generated we should switch to Benders cuts.

(default = 5000)

**Gomory_cuts (integer):** Frequency (in terms of nodes) for generating Gomory cuts in branch-and-cut.

See option 2mir_cuts for a detailed description.

(default = -5)
**hessian_approximation (string):** Indicates what Hessian information is to be used.

This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.

(default = exact)
- exact Use second derivatives provided by the NLP.
- limited-memory Perform a limited-memory quasi-Newton approximation

**hessian_approximation_space (string):** Indicates in which subspace the Hessian information is to be approximated.

(default = nonlinear-variables)
- all-variables in space of all variables (without slacks)
- nonlinear-variables only in space of nonlinear variables.

**heuristic_dive_fractional (string):** if yes runs the Dive Fractional heuristic

(default = no)
- no
- yes

**heuristic_dive_MIP_fractional (string):** if yes runs the Dive MIP Fractional heuristic

(default = no)
- no
- yes

**heuristic_dive_MIP_vectorLength (string):** if yes runs the Dive MIP VectorLength heuristic

(default = no)
- no
- yes

**heuristic_dive_vectorLength (string):** if yes runs the Dive VectorLength heuristic

(default = no)
- no
- yes

**heuristic_feasibility_pump (string):** whether the heuristic feasibility pump should be used

(default = no)
- no
- yes

**heuristic_RINS (string):** if yes runs the RINS heuristic

(default = no)
- no
- yes

**honor_original_bounds (string):** Indicates whether final points should be projected into original bounds.

Ipopt might relax the bounds during the optimization (see, e.g., option “bound_relax_factor”). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization.

(default = yes)
- no Leave final point unchanged
- yes Project final point back into original bounds
inf_pr_output (string): Determines what value is printed in the ‘inf_pr’ output column.

Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice “internal” prints out the constraint violation of this formulation. With “original” the true constraint violation in the original NLP is printed.

(default = original)

   internal max-norm of violation of internal equality constraints
   original maximal constraint violation in original NLP

integer_tolerance (real): Set integer tolerance.

Any number within that value of an integer is considered integer.

(default = 1e-06)

iteration_limit (integer): Set the cumulative maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound.

   value 0 deactivates option.

(default = maxint)

jacobian_regularization_exponent (real): Exponent for mu in the regularization for rank-deficient constraint Jacobians.

(This is kappa_c in the implementation paper.)

(default = 0.25)

jacobian_regularization_value (real): Size of the regularization for rank-deficient constraint Jacobians.

(This is bar delta_c in the implementation paper.)

(default = 1e-08)

jac_c_constant (string): Indicates whether all equality constraints are linear

Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.

(default = no)

   no Don’t assume that all equality constraints are linear
   yes Assume that equality constraints Jacobian are constant

jac_d_constant (string): Indicates whether all inequality constraints are linear

Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

(default = no)

   no Don’t assume that all inequality constraints are linear
   yes Assume that equality constraints Jacobian are constant

kappa_d (real): Weight for linear damping term (to handle one-sided bounds).

(see Section 3.7 in implementation paper.)

(default = 1e-05)

kappa_sigma (real): Factor limiting the deviation of dual variables from primal estimates.

If the dual variables deviate from their primal estimates, a correction is performed. (See Eqn. (16) in the implementation paper.) Setting the value to less than 1 disables the correction.

(default = 1e+10)

kappa_soc (real): Factor in the sufficient reduction rule for second order correction.

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. (See Step A-5.9 of Algorithm A in the implementation paper.)
least_square_init_duals (string): Least square initialization of all dual variables

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn’t know anything about the starting point, or for solving an LP or QP. This overwrites option "bound_mult_init_method".

(default = no)

- no use bound_mult_init_val and least-square equality constraint multipliers
- yes overwrite user-provided point with least-square estimates

least_square_init_primal (string): Least square initialization of the primal variables

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s), to fit the linearized equality and inequality constraints. This might be useful if the user doesn’t know anything about the starting point, or for solving an LP or QP.

(default = no)

- no take user-provided point
- yes overwrite user-provided point with least-square estimates

lift_and_project_cuts (integer): Frequency (in terms of nodes) for generating lift-and-project cuts in branch-and-cut

See option 2mir_cuts for a detailed description.

(default = 0)

limited_memory_aug_solver (string): Strategy for solving the augmented system for low-rank Hessian.

(default = sherman-morrison)

- extended use an extended augmented system
- sherman-morrison use Sherman-Morrison formula

limited_memory_initialization (string): Initialization strategy for the limited memory quasi-Newton approximation.

Determines how the diagonal Matrix B_0 as the first term in the limited memory approximation should be computed.

(default = scalar1)

- constant sigma = limited_memory_init_val
- scalar1 sigma = s^\top Ty/s^\top Ts
- scalar2 sigma = y^\top Ty/s^\top Ty
- scalar3 arithmetic average of scalar1 and scalar2
- scalar4 geometric average of scalar1 and scalar2

limited_memory_init_val (real): Value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1)

limited_memory_init_val_max (real): Upper bound on value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1e+08)

limited_memory_init_val_min (real): Lower bound on value for B0 in low-rank update.
The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".
(default = $1e^{-08}$)

**limited_memory_max_history (integer)**: Maximum size of the history for the limited quasi-Newton Hessian approximation.

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.
(default = 6)

**limited_memory_max_skipping (integer)**: Threshold for successive iterations where update is skipped.

If the update is skipped more than this number of successive iterations, we quasi-Newton approximation is reset.
(default = 2)

**limited_memory_special_for_resto (string)**: Determines if the quasi-Newton updates should be special during the restoration phase.

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

(default = no)
- no use the same update as in regular iterations
- yes use the a special update during restoration phase

**limited_memory_update_type (string)**: Quasi-Newton update formula for the limited memory approximation.

Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

(default = bfgs)
- bfgs BFGS update (with skipping)
- sr1 SR1 (not working well)

**linear_scaling_on_demand (string)**: Flag indicating that linear scaling is only done if it seems required.

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.

(default = yes)
- no Always scale the linear system.
- yes Start using linear system scaling if solutions seem not good.

**linear_solver (string)**: Linear solver used for step computations.

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, the code must have been compiled with the linear solver you want to choose. Depending on your Ipopt installation, not all options are available.

(default = ma27)
- ma27 use the Harwell routine MA27
- ma57 use the Harwell routine MA57
- ma77 use the Harwell routine HSL_MA77
- ma86 use the Harwell routine HSL_MA86
- ma97 use the Harwell routine HSL_MA97
- mumps use MUMPS package
pardiso use the Pardiso package

**linear_system_scaling** *(string)*: Method for scaling the linear system.

Determined the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. By default, MC19 is only used if MA27 or MA57 are selected as linear solvers. This value is only available if Ipopt has been compiled with MC19.

(default = mc19)

- mc19 use the Harwell routine MC19
- none no scaling will be performed
- slack-based use the slack values

**line_search_method** *(string)*: Globalization method used in backtracking line search

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.

(default = filter)

- cg-penalty Chen-Goldfarb penalty function
- filter Filter method
- penalty Standard penalty function

**lp_log_level** *(integer)*: specify LP log level.

Set the level of output of the linear programming sub-solver in B-Hyb or B-QG: 0 - none, 1 - minimal, 2 - normal low, 3 - normal high, 4 - verbose

Range: [0, 4]

(default = 0)

**ma27.ignore_singularity** *(string)*: Enables MA27’s ability to solve a linear system even if the matrix is singular.

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt’s heuristic of small perturbation of the lower diagonal of the KKT matrix.

(default = no)

- no Don’t have MA27 solve singular systems
- yes Have MA27 solve singular systems

**ma27_la_init_factor** *(real)*: Real workspace memory for MA27.

The initial real workspace memory = la_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

(default = 5)

**ma27_liw_init_factor** *(real)*: Integer workspace memory for MA27.

The initial integer workspace memory = liw_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

(default = 5)

**ma27_meminc_factor** *(real)*: Increment factor for workspace size for MA27.

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor. This option is only available if Ipopt has been compiled with MA27.

(default = 2)
**ma27.pivtol** *(real)*: Pivot tolerance for the linear solver MA27.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA27.

Range: [0, 1]
(default = 1e−08)

**ma27.pivtolmax** *(real)*: Maximum pivot tolerance for the linear solver MA27.

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA27.

Range: [0, 1]
(default = 0.0001)

**ma27.skip_inertia_check** *(string)*: Always pretend inertia is correct.

Setting this option to ”yes” essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

(default = no)

no check inertia
yes skip inertia check

**ma28.pivtol** *(real)*: Pivot tolerance for linear solver MA28.

This is used when MA28 tries to find the dependent constraints.

Range: [0, 1]
(default = 0.01)

**ma57.automatic_scaling** *(string)*: Controls MA57 automatic scaling

This option controls the internal scaling option of MA57. For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

(default = no)

no Do not scale the linear system matrix
yes Scale the linear system matrix

**ma57.block.size** *(integer)*: Controls block size used by Level 3 BLAS in MA57BD

This is ICNTL(11) in MA57.

(default = 16)

**ma57.node_amalgamation** *(integer)*: Node amalgamation parameter

This is ICNTL(12) in MA57.

(default = 16)

**ma57.pivot_order** *(integer)*: Controls pivot order in MA57

This is ICNTL(6) in MA57.

Range: [0, 5]
(default = 5)

**ma57.pivtol** *(real)*: Pivot tolerance for the linear solver MA57.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA57.

Range: [0, 1]
(default = 1e−08)
**ma57_pivotmax (real)**: Maximum pivot tolerance for the linear solver MA57.

Ipopt may increase pivot as high as `ma57_pivotmax` to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.

Range: [0, 1]
(default = 0.0001)

**ma57_pre_alloc (real)**: Safety factor for work space memory allocation for the linear solver MA57.

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice. This option is only available if Ipopt has been compiled with MA57.

(default = 1.05)

**ma57_small_pivot_flag (integer)**: If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.

This is ICNTL(16) in MA57.

Range: [0, 1]
(default = 0)

**ma77_buffer_lpage (integer)**: Number of scalars per MA77 buffer page

Number of scalars per an in-core buffer in the out-of-core solver MA77. Must be at most `ma77_file_size`.

(default = 4096)

**ma77_buffer_npage (integer)**: Number of pages that make up MA77 buffer

Number of pages of size `buffer_lpage` that exist in-core for the out-of-core solver MA77.

(default = 1600)

**ma77_file_size (integer)**: Target size of each temporary file for MA77, scalars per type

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.

(default = 2097152)

**ma77_maxstore (integer)**: Maximum storage size for MA77 in-core mode

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.

(default = 0)

**ma77_nemin (integer)**: Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than `ma77_nemin` variables.

(default = 8)

**ma77_order (string)**: Controls type of ordering used by HSL_MA77

This option controls ordering for the solver HSL_MA77.

(default = metis)

- Use the HSL_MC68 approximate minimum degree algorithm
- Use the MeTiS nested dissection algorithm (if available)

**ma77_print_level (integer)**: Debug printing level for the linear solver MA77

(default = -1)

**ma77_small (real)**: Zero Pivot Threshold

Any pivot less than `ma77_small` is treated as zero.
ma77\_static (real): Static Pivoting Threshold

See MA77 documentation. Either ma77\_static=0.0 or ma77\_static>ma77\_small. ma77\_static=0.0 disables static pivoting.

(default = 0)

ma77\_u (real): Pivoting Threshold

See MA77 documentation.

Range: [0, 0.5]

(default = 1e-08)

ma77\_umax (real): Maximum Pivoting Threshold

Maximum value to which u will be increased to improve quality.

Range: [0, 0.5]

(default = 0.0001)

ma86\_nemin (integer): Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than ma86\_nemin variables.

(default = 32)

ma86\_order (string): Controls type of ordering used by HSL\_MA86

This option controls ordering for the solver HSL\_MA86.

(default = auto)

  amd Use the HSL\_MC68 approximate minimum degree algorithm
  auto Try both AMD and MeTiS, pick best
  metis Use the MeTiS nested dissection algorithm (if available)

ma86\_print\_level (integer): Debug printing level for the linear solver MA86

(default = -1)

ma86\_scaling (string): Controls scaling of matrix

This option controls scaling for the solver HSL\_MA86.

(default = mc64)

  mc64 Scale linear system matrix using MC64
  mc77 Scale linear system matrix using MC77 [1,3,0]
  none Do not scale the linear system matrix

ma86\_small (real): Zero Pivot Threshold

Any pivot less than ma86\_small is treated as zero.

(default = 1e-20)

ma86\_static (real): Static Pivoting Threshold

See MA86 documentation. Either ma86\_static=0.0 or ma86\_static>ma86\_small. ma86\_static=0.0 disables static pivoting.

(default = 0)

ma86\_u (real): Pivoting Threshold

See MA86 documentation.

Range: [0, 0.5]
\textbf{ma86\_umax (real):} Maximum Pivoting Threshold

Maximum value to which \( u \) will be increased to improve quality.

Range: \([0, 0.5]\)

(default = 0.0001)

\textbf{ma97\_nemin (integer):} Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than \( \text{ma97\_nemin} \) variables.

(default = 8)

\textbf{ma97\_order (string):} Controls type of ordering used by HSL\_MA97

(default = auto)

\begin{itemize}
  \item \texttt{amd} Use the HSL\_MC68 approximate minimum degree algorithm
  \item \texttt{auto} Use HSL\_MA97 heuristic to guess best of AMD and METIS
  \item \texttt{best} Try both AMD and METIS, pick best
  \item \texttt{matched- amd} Use the HSL\_MC80 matching based ordering with AMD
  \item \texttt{matched- auto} Use the HSL\_MC80 matching with heuristic choice of AMD or METIS
  \item \texttt{matched- metis} Use the HSL\_MC80 matching based ordering with METIS
  \item \texttt{metis} Use the METIS nested dissection algorithm
\end{itemize}

\textbf{ma97\_print\_level (integer):} Debug printing level for the linear solver MA97

(default = 0)

\textbf{ma97\_scaling (string):} Specifies strategy for scaling in HSL\_MA97 linear solver

(default = dynamic)

\begin{itemize}
  \item \texttt{dynamic} Dynamically select scaling according to rules specified by \texttt{ma97\_scalingX} and \texttt{ma97\_switchX} options.
  \item \texttt{mc30} Scale all linear system matrices using MC30
  \item \texttt{mc64} Scale all linear system matrices using MC64
  \item \texttt{mc77} Scale all linear system matrices using MC77 \([1,3,0]\)
  \item \texttt{none} Do not scale the linear system matrix
\end{itemize}

\textbf{ma97\_scaling1 (string):} First scaling.

If \texttt{ma97\_scaling=dynamic}, this scaling is used according to the trigger \texttt{ma97\_switch1}. If \texttt{ma97\_switch2} is triggered it is disabled.

(default = \texttt{mc64})

\begin{itemize}
  \item \texttt{mc30} Scale linear system matrix using MC30
  \item \texttt{mc64} Scale linear system matrix using MC64
  \item \texttt{mc77} Scale linear system matrix using MC77 \([1,3,0]\)
  \item \texttt{none} No scaling
\end{itemize}

\textbf{ma97\_scaling2 (string):} Second scaling.

If \texttt{ma97\_scaling=dynamic}, this scaling is used according to the trigger \texttt{ma97\_switch2}. If \texttt{ma97\_switch3} is triggered it is disabled.

(default = \texttt{mc64})

\begin{itemize}
  \item \texttt{mc30} Scale linear system matrix using MC30
  \item \texttt{mc64} Scale linear system matrix using MC64
mc77 Scale linear system matrix using MC77 [1,3,0]
none No scaling

ma97.scaling3 (string): Third scaling.
   If ma97.scaling=dynamic, this scaling is used according to the trigger ma97.switch3.
   (default = mc64)
   mc30 Scale linear system matrix using MC30
   mc64 Scale linear system matrix using MC64
   mc77 Scale linear system matrix using MC77 [1,3,0]
   none No scaling

ma97.small (real): Zero Pivot Threshold
   Any pivot less than ma97.small is treated as zero.
   (default = 1e-20)

ma97.solve_blas3 (string): Controls if blas2 or blas3 routines are used for solve
   (default = no)
   no Use BLAS2 (faster, some implementations bit incompatible)
   yes Use BLAS3 (slower)

ma97.switch1 (string): First switch, determine when ma97.scaling1 is enabled.
   If ma97.scaling=dynamic, ma97.scaling1 is enabled according to this condition. If ma97.switch2 occurs this
   option is henceforth ignored.
   (default = od_hd_reuse)
   at_start Scaling to be used from the very start.
   at_start_reuse Scaling to be used on first iteration, then reused thereafter.
   high_delay Scaling to be used after more than 0.05*n delays are present
   high_delay_reuse Scaling to be used only when previous itr created more that 0.05*n additional
   delays, otherwise reuse scaling from previous itr
   never Scaling is never enabled.
   od_hd Combination of on_demand and high_delay
   od_hd_reuse Combination of on_demand_reuse and high_delay_reuse
   on_demand Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has
   failed).
   on_demand_reuse As on_demand, but reuse scaling from previous itr

ma97.switch2 (string): Second switch, determine when ma97.scaling2 is enabled.
   If ma97.scaling=dynamic, ma97.scaling2 is enabled according to this condition. If ma97.switch3 occurs this
   option is henceforth ignored.
   (default = never)
   at_start Scaling to be used from the very start.
   at_start_reuse Scaling to be used on first iteration, then reused thereafter.
   high_delay Scaling to be used after more than 0.05*n delays are present
   high_delay_reuse Scaling to be used only when previous itr created more that 0.05*n additional
   delays, otherwise reuse scaling from previous itr
   never Scaling is never enabled.
od_hd Combination of on_demand and high_delay
od_hd_reuse Combination of on_demand_reuse and high_delay_reuse
on_demand Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse As on_demand, but reuse scaling from previous itr

ma97_switch3 (string): Third switch, determine when ma97_scaling3 is enabled.
If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.
(default = never)
  at_start Scaling to be used from the very start.
at_start_reuse Scaling to be used on first iteration, then reused thereafter.
high_delay Scaling to be used after more than 0.05*n delays are present
high_delay_reuse Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
never Scaling is never enabled.
od_hd Combination of on_demand and high_delay
od_hd_reuse Combination of on_demand_reuse and high_delay_reuse
on_demand Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse As on_demand, but reuse scaling from previous itr

ma97_u (real): Pivoting Threshold
See MA97 documentation.
Range: [0, 0.5]
(default = 1e-08)

ma97_umax (real): Maximum Pivoting Threshold
See MA97 documentation.
Range: [0, 0.5]
(default = 0.0001)

maxmin_crit_have_sol (real): Weight towards minimum in of lower and upper branching estimates when a solution has been found.
Range: [0, 1]
(default = 0.1)

maxmin_crit_no_sol (real): Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.
Range: [0, 1]
(default = 0.7)

max_consecutive_failures (integer): (temporarily removed) Number n of consecutive unsolved problems before aborting a branch of the tree.
When n > 0, continue exploring a branch of the tree until n consecutive problems in the branch are unsolved (we call unsolved a problem for which Ipopt can not guarantee optimality within the specified tolerances).
(default = 10)

max_consecutive_infeasible (integer): Number of consecutive infeasible subproblems before aborting a branch.
Will continue exploring a branch of the tree until ”max_consecutive_infeasible” consecutive problems are locally
infeasible by the NLP sub-solver.

(default = 0)

**max_cpu_time (real):** Maximum number of CPU seconds.

A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding error message.

(default = 1e+06)

**max_filter_resets (integer):** Maximal allowed number of filter resets

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place.

(default = 5)

**max_hessian_perturbation (real):** Maximum value of regularization parameter for handling negative curvature.

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. (This is \( \delta_w^{\max} \) in the implementation paper.)

(default = 1e+20)

**max_iter (integer):** Maximum number of iterations.

The algorithm terminates with an error message if the number of iterations exceeded this number.

(default = 3000)

**max_random_point_radius (real):** Set max value \( r \) for coordinate of a random point.

When picking a random point, coordinate \( i \) will be in the interval \([\min(\max(l,-r),u-r), \max(\min(u,r),l+r)]\) (where \( l \) is the lower bound for the variable and \( u \) is its upper bound)

(default = 100000)

**max_refinement_steps (integer):** Maximum number of iterative refinement steps per linear system solve.

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps.

(default = 10)

**max_resto_iter (integer):** Maximum number of successive iterations in restoration phase.

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number.

(default = 3000000)

**max_soc (integer):** Maximum number of second order correction trial steps at each iteration.

Choosing 0 disables the second order corrections. (This is \( p^{\max} \) of Step A-5.9 of Algorithm A in the implementation paper.)

(default = 4)

**max_soft_resto_iters (integer):** Maximum number of iterations performed successively in soft restoration phase.

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called.

(default = 10)

**mehrotra_algorithm (string):** Indicates if we want to do Mehrotra’s algorithm.
If set to yes, Ipopt runs as Mehrotra’s predictor-corrector algorithm. This works usually very well for LPs and convex QPs. This automatically disables the line search, and chooses the (unglobalized) adaptive mu strategy with the "probing" oracle, and uses "corrector_type=affine" without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of "bound_push", "bound_frac", and "bound_mult_init_val" are set more aggressive, and sets "alpha_for_y=bound_mult".

(default = no)

no Do the usual Ipopt algorithm.

yes Do Mehrotra’s predictor-corrector algorithm.

**milp_log_level (integer):** specify MILP solver log level.

Set the level of output of the MILP subsolver in OA: 0 - none, 1 - minimal, 2

- normal low, 3 - normal high

Range: [0, 4]

(default = 0)

**milp_solver (string):** Choose the subsolver to solve MILP sub-problems in OA decompositions.

To use Cplex, a valid license is required.

(default = Cbc,D)

- cbc_d Coin Branch and Cut with its default
- cbc_par Coin Branch and Cut with passed parameters
- cplex IBM Cplex

**milp_strategy (string):** Choose a strategy for MILPs.

(default = solve_to_optimality)

- find_good_sol Stop sub milps when a solution improving the incumbent is found
- solve_to_optimality Solve MILPs to optimality

**min_hessian_perturbation (real):** Smallest perturbation of the Hessian block.

The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. (This is delta_w^\text{min} in implementation paper.)

(default = 1e-20)

**min_number_strong_branch (integer):** Sets minimum number of variables for strong branching (overriding trust)

(default = 0)

**min_refinement_steps (integer):** Minimum number of iterative refinement steps per linear system solve.

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.)

(default = 1)

**mir_cuts (integer):** Frequency (in terms of nodes) for generating MIR cuts in branch-and-cut

See option 2mir_cuts for a detailed description.

(default = -5)

**mumps_dep_tol (real):** Pivot threshold for detection of linearly dependent constraints in MUMPS.

When MUMPS is used to determine linearly dependent constraints, this is determines the threshold for a pivot to be considered zero. This is CNTL(3) in MUMPS.

(default = 0)

**mumps_mem_percent (integer):** Percentage increase in the estimated working space for MUMPS.
In MUMPS when significant extra fill-in is caused by numerical pivoting, larger values of mumps_mem_percent may help use the workspace more efficiently. On the other hand, if memory requirements are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

(default = 1000)

**mumps_piermuting_scaling** (*integer*): Controls permuting and scaling in MUMPS

This is ICNTL(6) in MUMPS.

Range: [0, 7]

(default = 7)

**mumps_pivot_order** (*integer*): Controls pivot order in MUMPS

This is ICNTL(7) in MUMPS.

Range: [0, 7]

(default = 7)

**mumps_pivtol** (*real*): Pivot tolerance for the linear solver MUMPS.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

(default = 1e-06)

**mumps_pivtolmax** (*real*): Maximum pivot tolerance for the linear solver MUMPS.

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

(default = 0.1)

**mumps_scaling** (*integer*): Controls scaling in MUMPS

This is ICNTL(8) in MUMPS.

Range: [-2, 77]

(default = 77)

**mu_allow_fast_monotone_decrease** (*string*): Allow skipping of barrier problem if barrier test is already met.

If set to "no", the algorithm enforces at least one iteration per barrier problem, even if the barrier test is already met for the updated barrier parameter.

(default = yes)

no Take at least one iteration per barrier problem

yes Allow fast decrease of mu if barrier test it met

**mu_init** (*real*): Initial value for the barrier parameter.

This option determines the initial value for the barrier parameter (mu). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if ”mu_strategy” is chosen as ”monotone”)

(default = 0.1)

**mu_linear_decrease_factor** (*real*): Determines linear decrease rate of barrier parameter.

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of mu∗”mu_linear_decrease_factor” and mu∧”superlinear_decrease_power”. (This is kappa_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: [0, 1]
mu_max (real): Maximum value for barrier parameter.
This option specifies an upper bound on the barrier parameter in the adaptive mu selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option “mu_strategy” is chosen as “adaptive”.)
(default = 0.2)

mu_max_fact (real): Factor for initialization of maximum value for barrier parameter.
This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option “mu_strategy” is chosen as “adaptive”.)
(default = 100000)

mu_min (real): Minimum value for barrier parameter.
This option specifies the lower bound on the barrier parameter in the adaptive mu selection mode. By default, it is set to the minimum of 1e-11 and min(“tol”,’compl_inf_tol’)/(“barrier_tol_factor”+1), which should be a reasonable value. (Only used if option “mu_strategy” is chosen as “adaptive”).
(default = 1e-11)

mu_oracle (string): Oracle for a new barrier parameter in the adaptive strategy.
Determines how a new barrier parameter is computed in each “free-mode” iteration of the adaptive barrier parameter strategy. (Only considered if “adaptive” is selected for option “mu_strategy”).
(default = probing)

   loqo LOQO’s centrality rule
   probing Mehrotra’s probing heuristic
   quality-function minimize a quality function

mu_strategy (string): Update strategy for barrier parameter.
Determines which barrier parameter update strategy is to be used.
(default = adaptive)

   adaptive use the adaptive update strategy
   monotone use the monotone (Fiacco-McCormick) strategy

mu_superlinear_decrease_power (real): Determines superlinear decrease rate of barrier parameter.
For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of mu*”mu_linear_decrease_factor” and mu^”superlinear_decrease_power”. (This is theta_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.
Range: [1, 2]
(default = 1.5)

mu_target (real): Desired value of complementarity.
Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of “mu_target” specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.
(default = 0)

neg_curv_test_tol (real): Tolerance for heuristic to ignore wrong inertia.
If positive, incorrect inertia in the augmented system is ignored, and we test if the direction is a direction of positive curvature. This tolerance determines when the direction is considered to be sufficiently positive.
**nlp_failure_behavior (string):** Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).

If set to "fathom", the algorithm will fathom the node when Ipopt fails to find a solution to the nlp at that node within the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal.

(default = stop)

- fathom: Continue when failure happens.
- stop: Stop when failure happens.

**nlp_log_at_root (integer):** specify a different log level for root relaxation.

Range: \([0, 12]\)

(default = 5)

**nlp_log_level (integer):** specify NLP solver interface log level (independent from ipopt print_level).

Set the level of output of the OsiTMINLPInterface : 0 - none, 1 - normal, 2 - verbose

Range: \([0, 2]\)

(default = 1)

**nlp_scaling_constr_target_gradient (real):** Target value for constraint function gradient size.

If a positive number is chosen, the scaling factor the constraint functions is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the constraint functions.

(default = 0)

**nlp_scaling_max_gradient (real):** Maximum gradient after NLP scaling.

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is g_max in Section 3.8 of the implementation paper.) Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

(default = 100)

**nlp_scaling_method (string):** Select the technique used for scaling the NLP.

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP. If you are using AMPL, they can be specified through suffixes ("scaling factor")

(default = gradient-based)

- equilibration-based: scale the problem so that first derivatives are of order 1 at random points (only available with MC19)
- gradient-based: scale the problem so the maximum gradient at the starting point is scaling_max_gradient
- none: no problem scaling will be performed

**nlp_scaling_min_value (real):** Minimum value of gradient-based scaling values.

This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

(default = 1e-08)

**nlp_scaling_obj_target_gradient (real):** Target value for objective function gradient size.

If a positive number is chosen, the scaling factor the objective function is computed so that the gradient has
the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the objective function.
(default = 0)

**nlp_solves_per_depth (real)**: Set average number of nodes in the tree at which NLP relaxations are solved in B-Hyb for each depth.
(default = 1e+100)

**nlp_solve_frequency (integer)**: Specify the frequency (in terms of nodes) at which NLP relaxations are solved in B-Hyb.
A frequency of 0 amounts to to never solve the NLP relaxation.
(default = 10)

**nlp_solve_max_depth (integer)**: Set maximum depth in the tree at which NLP relaxations are solved in B-Hyb.
A depth of 0 amounts to to never solve the NLP relaxation.
(default = 10)

**node_comparison (string)**: Choose the node selection strategy.
Choose the strategy for selecting the next node to be processed.
(default = best-bound)

- **best-bound**: choose node with the smallest bound,
- **best-guess**: choose node with smallest guessed integer solution
- **breadth-first**: Perform breadth first search,
- **depth-first**: Perform depth first search,
- **dynamic**: Cbc dynamic strategy (starts with a depth first search and turn to best bound after 3 integer feasible solutions have been found).

**node_limit (integer)**: Set the maximum number of nodes explored in the branch-and-bound search.
(default = maxint)

**number_before_trust (integer)**: Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching.
A value of 0 disables pseudo costs.
(default = 8)

**number_before_trust_list (integer)**: Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list.
The default value is that of ”number_before_trust”
(default = 0)

**number_cpx_threads (integer)**: Set number of threads to use with cplex.
(refer to CPLEX documentation)
(default = 0)

**number_look_ahead (integer)**: Sets limit of look-ahead strong-branching trials
(default = 0)

**number_strong_branch (integer)**: Choose the maximum number of variables considered for strong branching.
Set the number of variables on which to do strong branching.
(default = 20)

**number_strong_branch_root (integer)**: Maximum number of variables considered for strong branching in root node.
(default = maxint)
num_cut_passes (integer): Set the maximum number of cut passes at regular nodes of the branch-and-cut.
  (default = 1)

num_cut_passes_at_root (integer): Set the maximum number of cut passes at regular nodes of the branch-and-cut.
  (default = 20)

num_iterations_suspect (integer): Number of iterations over which a node is considered 'suspect' (for debugging purposes only, see detailed documentation).
  When the number of iterations to solve a node is above this number, the subproblem at this node is considered to be suspect and it will be written into a file (set to -1 to deactivate this).
  (default = -1)

num_linear_variables (integer): Number of linear variables
  When the Hessian is approximated, it is assumed that the first num_linear_variables variables are linear. The Hessian is then not approximated in this space. If the get_number_of_nonlinear_variables method in the TNLP is implemented, this option is ignored.
  (default = 0)

num_resolve_at_infeasibles (integer): Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point.
  The algorithm will solve all the infeasible nodes with $k$ different random starting points and will keep the best local optimum found.
  (default = 0)

num_resolve_at_node (integer): Number $k$ of tries to resolve a node (other than the root) of the tree with different starting point.
  The algorithm will solve all the nodes with $k$ different random starting points and will keep the best local optimum found.
  (default = 0)

num_resolve_at_root (integer): Number $k$ of tries to resolve the root node with different starting points.
  The algorithm will solve the root node with $k$ random starting points and will keep the best local optimum found.
  (default = 0)

num_retry_unsolved_random_point (integer): Number $k$ of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).
  When Ipopt fails to solve a continuous NLP sub-problem, if $k > 0$, the algorithm will try again to solve the failed NLP with $k$ new randomly chosen starting points or until the problem is solved with success.
  (default = 0)

nu_inc (real): Increment of the penalty parameter.
  (default = 0.0001)

nu_init (real): Initial value of the penalty parameter.
  (default = 1e-06)

oa_cuts_log_level (integer): level of log when generating OA cuts.
  0: outputs nothing, 1: when a cut is generated, its violation and index of row from which it originates, 2: always output violation of the cut. 3: output generated cuts incidence vectors.
  (default = 0)

oa_cuts_scope (string): Specify if OA cuts added are to be set globally or locally valid
BONMIN and BONMINH

(global)

global Cuts are treated as globally valid
local Cuts are treated as locally valid

**oa_decomposition** *(string)*: If yes do initial OA decomposition
(default = no)

no
yes

**oa_log_frequency** *(real)*: display an update on lower and upper bounds in OA every n seconds
(default = 100)

**oa_log_level** *(integer)*: specify OA iterations log level.

Set the level of output of OA decomposition solver: 0 - none, 1 - normal, 2 - verbose
Range: [0, 2]
(default = 1)

**oa_rhs_relax** *(real)*: Value by which to relax OA cut

RHS of OA constraints will be relaxed by this amount times the absolute value of the initial rhs if it is >= 1
(otherwise by this amount).
(default = 1e-08)

**obj_max_inc** *(real)*: Determines the upper bound on the acceptable increase of barrier objective function.

Trial points are rejected if they lead to an increase in the barrier objective function by more than obj_max_inc orders of magnitude.
(default = 5)

**pardiso_matching_strategy** *(string)*: Matching strategy to be used by Pardiso

This is IPAR(13) in Pardiso manual.

(default = complete+2x2)

- **complete** Match complete (IPAR(13)=1)
- **complete+2x2** Match complete+2x2 (IPAR(13)=2)
- **constraints** Match constraints (IPAR(13)=3)

**pardiso_max_iterative_refinement_steps** *(integer)*: Limit on number of iterative refinement steps.

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

(default = 1)

**pardiso_msglvl** *(integer)*: Pardiso message level

This determines the amount of analysis output from the Pardiso solver. This is MSGLVL in the Pardiso manual.

(default = 0)

**pardiso_order** *(string)*: Controls the fill-in reduction ordering algorithm for the input matrix.

(default = metis)

- **amd** minimum degree algorithm
- **metis** MeTiS nested dissection algorithm
onen undocumented

pmeti$ parallel (OpenMP) version of MeTiS nested dissection algorithm

**pardiso redo_symbolic_fact_only_if_inertia_wrong (string):** Toggle for handling case when elements were perturbed by Pardiso.

(default = no)

- no Always redo symbolic factorization when elements were perturbed
- yes Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

**pardiso_repeated_perturbation_means_singular (string):** Interpretation of perturbed elements.

(default = no)

- no Don’t assume that matrix is singular if elements were perturbed after recent symbolic factorization
- yes Assume that matrix is singular if elements were perturbed after recent symbolic factorization

**pardiso_skip_inertia_check (string):** Always pretend inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

(default = no)

- no check inertia
- yes skip inertia check

**perturb_always_cd (string):** Active permanent perturbation of constraint linearization.

This options makes the delta.c and delta.d perturbation be used for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.

(default = no)

- no perturbation only used when required
- yes always use perturbation

**perturb_dec_fact (real):** Decrease factor for x-s perturbation.

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. (This is kappa.w^- in the implementation paper.)

Range: $[0, 1]$

(default = 0.333333)

**perturb_inc_fact (real):** Increase factor for x-s perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. (This is kappa.w^+ in the implementation paper.)

(default = 8)

**perturb_inc_fact_first (real):** Increase factor for x-s perturbation for very first perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for for the first perturbation than that used for the remaining perturbations. (This is bar_kappa.w^+ in the implementation paper.)

(default = 100)

**print_eval_error (string):** Switch to enable printing information about function evaluation errors into the GAMS listing file.

(default = yes)

- no
- yes
**print_frequency_iter (integer)**: Determines at which iteration frequency the summarizing iteration output line should be printed.

Summarizing iteration output is printed every print_frequency_iter iterations, if at least print_frequency_time seconds have passed since last output.

(default = 1)

**print_frequency_time (real)**: Determines at which time frequency the summarizing iteration output line should be printed.

Summarizing iteration output is printed if at least print_frequency_time seconds have passed since last output and the iteration number is a multiple of print_frequency_iter.

(default = 0.5)

**print_funceval_statistics (string)**: Switch to enable printing statistics on number of evaluations of GAMS functions/gradient/Hessian.

(default = no)

no
yes

**print_info_string (string)**: Enables printing of additional info string at end of iteration output.

This string contains some insider information about the current iteration. For details, look for "Diagnostic Tags" in the Ipopt documentation.

(default = no)

no don’t print string
yes print string at end of each iteration output

**print_level (integer)**: Output verbosity level.

Sets the default verbosity level for console output. The larger this value the more detailed is the output.

Range: [0, 12]

(default = 0)

**print_timing_statistics (string)**: Switch to print timing statistics.

If selected, the program will print the CPU usage (user time) for selected tasks.

(default = no)

no don’t print statistics
yes print all timing statistics

**pump_for_minlp (string)**: whether to run the feasibility pump heuristic for MINLP

(default = no)

no
yes

**quality_function_balancing_term (string)**: The balancing term included in the quality function for centrality.

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. (Only used if option "mu_oracle" is set to "quality-function").

(default = none)

cubic Max(0,Max(dual_inf,primal_inf)-compl)^3
none no balancing term is added

**quality_function_centrality (string)**: The penalty term for centrality that is included in quality function.

This determines whether a term is added to the quality function to penalize deviation from centrality with respect
to complementarity. The complementarity measure here is the xi in the Loqo update rule. (Only used if option "mu_oracle" is set to "quality-function".)

(default = none)

- cubed-reciprocal complementarity * the reciprocal of the centrality measure cubed
- log complementarity * the log of the centrality measure
- none no penalty term is added
- reciprocal complementarity * the reciprocal of the centrality measure

**quality_function_max_section_steps (integer):** Maximum number of search steps during direct search procedure determining the optimal centering parameter.

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function").

(default = 8)

**quality_function_norm_type (string):** Norm used for components of the quality function.

(Only used if option "mu_oracle" is set to "quality-function").

(default = 2-norm-squared)

- 1-norm use the 1-norm (abs sum)
- 2-norm use 2-norm
- 2-norm-squared use the 2-norm squared (sum of squares)
- max-norm use the infinity norm (max)

**quality_function_section_qf_tol (real):** Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function").

Range: [0, 1]

(default = 0)

**quality_function_section_sigma_tol (real):** Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function").

Range: [0, 1]

(default = 0.01)

**random_generator_seed (integer):** Set seed for random number generator (a value of -1 sets seeds to time since Epoch).

(default = 0)

**random_point_perturbation_interval (real):** Amount by which starting point is perturbed when choosing to pick random point by perturbing starting point

(default = 1)

**random_point_type (string):** method to choose a random starting point

(default = Jon)

- andreas perturb the starting point of the problem within a prescribed interval
- claudia perturb the starting point using the perturbation radius suffix information
- jon Choose random point uniformly between the bounds

**read_solution_file (string):** Read a file with the optimal solution to test if algorithms cuts it.
For Debugging purposes only.
(default = no)

no
yes

recalc_y (string): Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than recalc_y_feas_tol. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default.

(default = no)

no use the Newton step to update the multipliers
yes use least-square multiplier estimates

recalc_y_feas_tol (real): Feasibility threshold for recomputation of multipliers.

If recalc_y is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.

(default = 1e-06)

reduce_and_split_cuts (integer): Frequency (in terms of nodes) for generating reduce-and-split cuts in branch-and-cut

See option 2mir_cuts for a detailed description.

(default = 0)

replace_bounds (string): Indicates if all variable bounds should be replaced by inequality constraints

This option must be set for the inexact algorithm

(default = no)

no leave bounds on variables
yes replace variable bounds by inequality constraints

required_infeasibility_reduction (real): Required reduction of infeasibility before leaving restoration phase.

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option.

Range: [0, 1]

(default = 0.1)

residual_improvement_factor (real): Minimal required reduction of residual test ratio in iterative refinement.

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted.

(default = 1)

residual_ratio_max (real): Iterative refinement tolerance

Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

(default = 1e-10)

residual_ratio_singular (real): Threshold for declaring linear system singular after failed iterative refinement.

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

(default = 1e-05)

resolve_on_small_infeasibility (real): If a locally infeasible problem is infeasible by less than this, resolve it with initial starting point.
resto_failure_feasibility_threshold (real): Threshold for primal infeasibility to declare failure of restoration phase.

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is 1e2*tol, where tol is the general termination tolerance.

(resto_feasibility_threshold = 0)

resto_penalty_parameter (real): Penalty parameter in the restoration phase objective function.

This is the parameter rho in equation (31a) in the Ipopt implementation paper.

(resto_penalty_parameter = 1000)

resto_proximity_weight (real): Weighting factor for the proximity term in restoration phase objective.

This determines how the parameter zeta in equation (29a) in the implementation paper is computed. Zeta here is resto_proximity_weight*sqrt(mu), where mu is the current barrier parameter.

(resto_proximity_weight = 1)

rho (real): Value in penalty parameter update formula.

Range: [0, 1]

(rho = 0.1)

second_perc_for_cutoff_decr (real): The percentage used when the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically.

(second_perc_for_cutoff_decr = -0.05)

setup_pseudo_frac (real): Proportion of strong branching list that has to be taken from most-integer-infeasible list.

Range: [0, 1]

(setup_pseudo_frac = 0.5)

sigma_max (real): Maximum value of the centering parameter.

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

(sigma_max = 100)

sigma_min (real): Minimum value of the centering parameter.

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

(sigma_min = 1e-06)

skip_corr_if_neg_curv (string): Skip the corrector step in negative curvature iteration (unsupported!).

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive".

(skip_corr_if_neg_curv = yes)

  no don’t skip

  yes skip

skip_corr_in_monotone_mode (string): Skip the corrector step during monotone barrier parameter mode (unsupported!).

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive".

(skip_corr_in_monotone_mode = yes)

  no don’t skip

  yes skip
**slack\_bound\_frac** (*real*): Desired minimum relative distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack\_bound\_push"). (This is kappa\_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

(default = 0.01)

**slack\_bound\_push** (*real*): Desired minimum absolute distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack\_bound\_frac"). (This is kappa\_1 in Section 3.6 of implementation paper.)

(default = 0.01)

**slack\_move** (*real*): Correction size for very small slacks.

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is mach\_eps$$^\{3/4\}$$. (See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.)

(default = 1.81899e-12)

**soft\_resto\_p\_derror\_reduction\_factor** (*real*): Required reduction in primal-dual error in the soft restoration phase.

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase.

(default = 0.9999)

**solution\_limit** (*integer*): Abort after that much integer feasible solution have been found by algorithm

value 0 deactivates option

(default = maxint)

**solvefinal** (*string*): Switch to disable solving MINLP with discrete variables fixed to solution values after solve.

If enabled, then the dual values from the resolved NLP are made available in GAMS.

(default = yes)

no

yes

**solvetrace** (*string*): Name of file for writing solving progress information.

**solvetracenodefreq** (*integer*): Frequency in number of nodes for writing solving progress information.

Giving 0 disables writing of N-lines to trace file

(default = 100)

**solvetracetimefreq** (*real*): Frequency in seconds for writing solving progress information.

Giving 0.0 disables writing of T-lines to trace file

(default = 5)

**start\_with\_resto** (*string*): Tells algorithm to switch to restoration phase in first iteration.

Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration.

If the initial point is feasible, the algorithm will abort with a failure.

(default = no)

no don’t force start in restoration phase

yes force start in restoration phase

**s\_max** (*real*): Scaling threshold for the NLP error.
s\_phi (\textit{real}): Exponent for linear barrier function model in the switching rule.
(See Eqn. (19) in the implementation paper.)
(default = 2.3)

\textbf{tau\_min (\textit{real}):} Lower bound on fraction-to-the-boundary parameter tau.
(This is tau\_min in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.
Range: [0, 1]
(default = 0.99)

\textbf{theta\_max\_fact (\textit{real}):} Determines upper bound for constraint violation in the filter.
The algorithmic parameter theta\_max is determined as theta\_max\_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta\_max is unacceptable to the filter (see Eqn. (21) in the implementation paper).
(default = 10000)

\textbf{theta\_min\_fact (\textit{real}):} Determines constraint violation threshold in the switching rule.
The algorithmic parameter theta\_min is determined as theta\_min\_fact times the maximum of 1 and the constraint violation at initial point. The switching rules treats an iteration as an h-type iteration whenever the current constraint violation is larger than theta\_min (see paragraph before Eqn. (19) in the implementation paper).
(default = 0.0001)

\textbf{theta\_max (\textit{real}):} Value for tiny element in OA cut
We will remove "cleanly" (by relaxing cut) an element lower than this.
(default = 1e-08)

\textbf{tiny\_step\_tol (\textit{real}):} Tolerance for detecting numerically insignificant steps.
If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.
(default = 2.22045e-15)

\textbf{tiny\_step\_y\_tol (\textit{real}):} Tolerance for quitting because of numerically insignificant steps.
If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny\_step\_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.
(default = 0.01)

\textbf{tol (\textit{real}):} Desired convergence tolerance (relative).
Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual\_inf\_tol", "constr\_viol\_tol", and "compl\_inf\_tol" are met. (This is epsilon\_tol in Eqn. (6) in implementation paper). See also "acceptable\_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.
(default = 1e-08)

**tree_search_strategy** (string): Pick a strategy for traversing the tree

All strategies can be used in conjunction with any of the node comparison functions. Options which affect dfs-dive are max-backtracks-in-dive and max-dive-depth. The dfs-dive won’t work in a non-convex problem where objective does not decrease down branches.

(default = probed-dive)

dfs-dive Dive in the tree if possible doing a depth first search. Backtrack on leaves or when a prescribed depth is attained or when estimate of best possible integer feasible solution in subtree is worst than cutoff.

dfs-dive-dynamic Same as dfs-dive but once enough solution are found switch to best-bound and if too many nodes switch to depth-first.

dive Dive in the tree if possible, otherwise pick top node as sorted by the tree comparison function.

probed-dive Dive in the tree exploring two children before continuing the dive at each level.

top-node Always pick the top node as sorted by the node comparison function

**trust_strong_branching_for_pseudo_cost** (string): Whether or not to trust strong branching results for updating pseudo costs.

(default = yes)

no

yes

**variable_selection** (string): Chooses variable selection strategy

(default = strong-branching)

lp-strong-branching Perform strong branching with LP approximation

most-fractional Choose most fractional variable

nlp-strong-branching Perform strong branching with NLP approximation

osi-simple Osi method to do simple branching

osi-strong Osi method to do strong branching

qp-strong-branching Perform strong branching with QP approximation

random Method to choose branching variable randomly

reliability-branching Use reliability branching

strong-branching Perform strong branching

**very_tiny_element** (real): Value for very tiny element in OA cut

Algorithm will take the risk of neglecting an element lower than this.

(default = 1e-17)

**warm_start** (string): Select the warm start method

This will affect the function getWarmStart(), and as a consequence the warm starting in the various algorithms.

(default = none)

fake_basis builds fake basis, useful for cut management in Cbc (warm start is the same as in none)

interior_point Warm start with an interior point of direct parent

none No warm start, just start NLPs from optimal solution of the root relaxation

optimum Warm start with direct parent optimum

**warm_start_bound_frac** (real): same as bound_frac for the regular initializer.
Range: $[0, 0.5]$

(default = 0.001)

**warm_start_bound_push** *(real):* same as bound_push for the regular initializer.

(default = 0.001)

**warm_start_init_point** *(string):* Warm-start for initial point

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)

(default = yes)

no do not use the warm start initialization

yes use the warm start initialization

**warm_start_mult_bound_push** *(real):* same as mult_bound_push for the regular initializer.

(default = 0.001)

**warm_start_mult_init_max** *(real):* Maximum initial value for the equality multipliers.

(default = 1e+06)

**warm_start_slack_bound_frac** *(real):* same as slack_bound_frac for the regular initializer.

Range: $[0, 0.5]$

(default = 0.001)

**warm_start_slack_bound_push** *(real):* same as slack_bound_push for the regular initializer.

(default = 0.001)

**watchdog_shortened_iter_trigger** *(integer):* Number of shortened iterations that trigger the watchdog.

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure.

(default = 10)

**watchdog_trial_iter_max** *(integer):* Maximum number of watchdog iterations.

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.

(default = 3)

**Bibliography**


CBC

Contents

1 Usage ................................................................. 631
2 List of Options ....................................................... 632
   2.1 General Options .................................................. 632
   2.2 LP Options ....................................................... 632
   2.3 MIP Options ....................................................... 632
   2.4 MIP Options for Cutting Plane Generators ...................... 633
   2.5 MIP Options for Heuristics ...................................... 634
3 Detailed Options Description ...................................... 634

CBC (COIN-OR Branch and Cut) is an open-source mixed integer programming solver working with the COIN-OR LP solver CLP and the COIN-OR Cut generator library Cgl. The code has been written primarily by John J. Forrest.

For more information we refer to the website of CBC, CGL, and CLP. Most of the CBC documentation in the section was copied from the help in the CBC standalone version.

The CBC link in GAMS supports continuous, binary, integer, semicontinuous, semiinteger variables, special ordered sets of type 1 and 2, and branching priorities (see chapter 17.1 of the GAMS User’s Guide).

1 Usage

The following statement can be used inside your GAMS program to specify using CBC

Option LP = CBC; { or MIP or RMIP }

The above statement should appear before the Solve statement. If CBC was specified as the default solver during GAMS installation, the above statement is not necessary.

For usage and syntax of solver options file, see Section The Solver Option File. Following is an example options file cbc.opt.

cuts root
perturbation off

It will cause CBC to use cut generators only in the root node and turns off the perturbation of the LP relaxation.

GAMS/CBC currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/CBC with BCH, please consider to use a GAMS system of version \( \leq 23.3 \).

The following GAMS parameters are currently supported by GAMS/CBC: reslim, iterlim, nodlim, optca, optcr, cheat, cutoff, and threads.
2 List of Options

There are many parameters which can affect the performance the CBCs Branch and Cut Algorithm. First just try with default settings and look carefully at the log file. Did cuts help? Did they take too long? Look at the output to see which cuts were effective and then do some tuning (see the option cuts). If the preprocessing reduced the size of the problem or strengthened many coefficients then it is probably wise to leave it on. Switch off heuristics which did not provide solutions. The other major area to look at is the search. Hopefully good solutions were obtained fairly early in the search so the important point is to select the best variable to branch on. See whether strong branching did a good job – or did it just take a lot of iterations? Adjust the options strongbranching and trustpseudocosts.

In the following, we summarize all available CBC options.

2.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>reslim</td>
<td>resource limit</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>special</td>
<td>options passed unseen to CBC</td>
<td></td>
</tr>
<tr>
<td>writemps</td>
<td>create MPS file for problem</td>
<td></td>
</tr>
</tbody>
</table>

2.2 LP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash</td>
<td>use crash method to get dual feasible</td>
<td>off</td>
</tr>
<tr>
<td>crossover</td>
<td>crossover to simplex algorithm after barrier</td>
<td>1</td>
</tr>
<tr>
<td>dualpiv</td>
<td>dual pivot choice algorithm</td>
<td>auto</td>
</tr>
<tr>
<td>idiotcrash</td>
<td>idiot crash</td>
<td>-1</td>
</tr>
<tr>
<td>iterlim</td>
<td>iteration limit</td>
<td>GAMS iterlim</td>
</tr>
<tr>
<td>maxfactor</td>
<td>maximum number of iterations between refactorizations</td>
<td>200</td>
</tr>
<tr>
<td>passpresolve</td>
<td>how many passes to do in presolve</td>
<td>5</td>
</tr>
<tr>
<td>perturbation</td>
<td>perturbation of problem</td>
<td>1</td>
</tr>
<tr>
<td>presolve</td>
<td>switch for initial presolve of LP</td>
<td>1</td>
</tr>
<tr>
<td>primalpivot</td>
<td>primal pivot choice algorithm</td>
<td>auto</td>
</tr>
<tr>
<td>randomseedclp</td>
<td>random seed for CLP</td>
<td>-1</td>
</tr>
<tr>
<td>scaling</td>
<td>scaling method</td>
<td>auto</td>
</tr>
<tr>
<td>sifting</td>
<td>synonym for sprint crash</td>
<td>-1</td>
</tr>
<tr>
<td>sprintcrash</td>
<td>sprint crash</td>
<td>-1</td>
</tr>
<tr>
<td>startalg</td>
<td>LP solver for root node</td>
<td>dual</td>
</tr>
<tr>
<td>tol_dual</td>
<td>dual feasibility tolerance</td>
<td>1e-7</td>
</tr>
<tr>
<td>tol_presolve</td>
<td>tolerance used in presolve</td>
<td>1e-8</td>
</tr>
<tr>
<td>tol_primal</td>
<td>primal feasibility tolerance</td>
<td>1e-7</td>
</tr>
</tbody>
</table>

2.3 MIP Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>coststrategy</td>
<td>how to use costs as priorities</td>
<td>off</td>
</tr>
<tr>
<td>cutoff</td>
<td>cutoff for objective function value</td>
<td>GAMS cutoff</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>cutoffconstrainer</td>
<td>whether to add a constraint from the objective function</td>
<td>0</td>
</tr>
<tr>
<td>dumpssolutions</td>
<td>name of solutions index gdx file for writing alternate solutions</td>
<td></td>
</tr>
<tr>
<td>extravariables</td>
<td>group together variables with same cost</td>
<td>0</td>
</tr>
<tr>
<td>increment</td>
<td>increment of cutoff when new incumbent</td>
<td>GAMS cheat</td>
</tr>
<tr>
<td>loglevel</td>
<td>CBC loglevel</td>
<td>1</td>
</tr>
<tr>
<td>maxsol</td>
<td>maximal number of solutions to store during search</td>
<td>100</td>
</tr>
<tr>
<td>mipstart</td>
<td>whether it should be tried to use the initial variable levels as initial MIP solution</td>
<td>0</td>
</tr>
<tr>
<td>multplerootpasses</td>
<td>runs multiple copies of the solver at the root node</td>
<td>0</td>
</tr>
<tr>
<td>nodelim</td>
<td>node limit</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td>nodestrategy</td>
<td>how to select nodes</td>
<td>fewest</td>
</tr>
<tr>
<td>nodlim</td>
<td>node limit</td>
<td>GAMS nodlim</td>
</tr>
<tr>
<td>optca</td>
<td>absolute stopping tolerance</td>
<td>GAMS optca</td>
</tr>
<tr>
<td>optcr</td>
<td>relative stopping tolerance</td>
<td>GAMS optcr</td>
</tr>
<tr>
<td>parallelmode</td>
<td>whether to run opportunistic or deterministic</td>
<td>deterministic</td>
</tr>
<tr>
<td>preprocess</td>
<td>integer presolve</td>
<td>on</td>
</tr>
<tr>
<td>printfrequency</td>
<td>frequency of status prints</td>
<td>0</td>
</tr>
<tr>
<td>randomseedcbc</td>
<td>random seed for CBC</td>
<td></td>
</tr>
<tr>
<td>sollim</td>
<td>limit on number of solutions</td>
<td>-1</td>
</tr>
<tr>
<td>solvefinal</td>
<td>final solve of MIP with fixed discrete variables</td>
<td>1</td>
</tr>
<tr>
<td>solvtrace</td>
<td>name of trace file for solving information</td>
<td></td>
</tr>
<tr>
<td>solvtracenodefreq</td>
<td>frequency in number of nodes for writing to solve trace file</td>
<td>100</td>
</tr>
<tr>
<td>solvtraceftimefreq</td>
<td>frequency in seconds for writing to solve trace file</td>
<td>5</td>
</tr>
<tr>
<td>strategy</td>
<td>switches on groups of features</td>
<td>1</td>
</tr>
<tr>
<td>strongbranching</td>
<td>strong branching</td>
<td>5</td>
</tr>
<tr>
<td>threads</td>
<td>number of threads to use</td>
<td>GAMS threads</td>
</tr>
<tr>
<td>tol_integer</td>
<td>tolerance for integrality</td>
<td>1e-6</td>
</tr>
<tr>
<td>trustpseudocosts</td>
<td>after howmany nodes we trust the pseudo costs</td>
<td>5</td>
</tr>
</tbody>
</table>

### 2.4 MIP Options for Cutting Plane Generators

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cliquecuts</td>
<td>Clique Cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>cutdepth</td>
<td>depth in tree at which cuts are applied</td>
<td>-1</td>
</tr>
<tr>
<td>cuts</td>
<td>global switch for cutgenerators</td>
<td>on</td>
</tr>
<tr>
<td>cut_passes_root</td>
<td>number of cut passes at root node</td>
<td>20 or 100</td>
</tr>
<tr>
<td>cut_passes_slow</td>
<td>number of cut passes for slow cut generators</td>
<td>10</td>
</tr>
<tr>
<td>cut_passes_tree</td>
<td>number of cut passes at nodes in the tree</td>
<td>1</td>
</tr>
<tr>
<td>flowcovercuts</td>
<td>Flow Cover Cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>gomorycuts</td>
<td>Gomory Cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>gomorycuts2</td>
<td>Gomory Cuts 2nd implementation</td>
<td>off</td>
</tr>
<tr>
<td>knapsackcuts</td>
<td>Knapsack Cover Cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>liftandprojectcuts</td>
<td>Lift and Project Cuts</td>
<td>off</td>
</tr>
<tr>
<td>mircuts</td>
<td>Mixed Integer Rounding Cuts</td>
<td>ifmove</td>
</tr>
<tr>
<td>probingcuts</td>
<td>Probing Cuts</td>
<td>ifmove</td>
</tr>
</tbody>
</table>
2.5 MIP Options for Heuristics

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>combinesolutions</td>
<td>combine solutions heuristic</td>
<td>1</td>
</tr>
<tr>
<td>dins</td>
<td>distance induced neighborhood search</td>
<td>0</td>
</tr>
<tr>
<td>divingcoefficient</td>
<td>coefficient diving heuristic</td>
<td>1</td>
</tr>
<tr>
<td>divingfractional</td>
<td>fractional diving heuristic</td>
<td>0</td>
</tr>
<tr>
<td>divingguided</td>
<td>guided diving heuristic</td>
<td>0</td>
</tr>
<tr>
<td>divinglinesearch</td>
<td>line search diving heuristic</td>
<td>0</td>
</tr>
<tr>
<td>divingpseudocost</td>
<td>pseudo cost diving heuristic</td>
<td>0</td>
</tr>
<tr>
<td>divingrandom</td>
<td>turns on random diving heuristic</td>
<td>0</td>
</tr>
<tr>
<td>divingletectorlength</td>
<td>vector length diving heuristic</td>
<td>0</td>
</tr>
<tr>
<td>feasumpump</td>
<td>feasibility pump</td>
<td>1</td>
</tr>
<tr>
<td>feasumpump_passes</td>
<td>number of feasibility passes</td>
<td>20</td>
</tr>
<tr>
<td>greedyheuristic</td>
<td>greedy heuristic</td>
<td>on</td>
</tr>
<tr>
<td>heuristics</td>
<td>global switch for heuristics</td>
<td>1</td>
</tr>
<tr>
<td>localtreesearch</td>
<td>local tree search heuristic</td>
<td>0</td>
</tr>
<tr>
<td>naiveheuristics</td>
<td>naive heuristics</td>
<td>0</td>
</tr>
<tr>
<td>pivotandfix</td>
<td>pivot and fix heuristic</td>
<td>0</td>
</tr>
<tr>
<td>proximitysearch</td>
<td>proximity search heuristic</td>
<td>0</td>
</tr>
<tr>
<td>randomizedrounding</td>
<td>randomized rounding heuristic</td>
<td>0</td>
</tr>
<tr>
<td>rens</td>
<td>relaxation enforced neighborhood search</td>
<td>0</td>
</tr>
<tr>
<td>rins</td>
<td>relaxed induced neighborhood search</td>
<td>0</td>
</tr>
<tr>
<td>roundingheuristic</td>
<td>rounding heuristic</td>
<td>1</td>
</tr>
<tr>
<td>vubheuristic</td>
<td>VUB heuristic</td>
<td></td>
</tr>
</tbody>
</table>

In the following, we give a detailed description of all available CBC options.

3 Detailed Options Description

cliquecuts (string): Clique Cuts

Determines whether and when CBC should try to generate clique cuts. See the option cuts for an explanation on the different values. Clique cuts are of the form “sum of a set of variables <= 1”. Reference: M. Eso, Parallel branch and cut for set partitioning, Cornell University, 1999.

(default = if move)

combinesolutions (integer): combine solutions heuristic

This parameter control the use of a heuristic which does branch and cut on the given problem by just using variables which have appeared in one or more solutions. It is obviously only tried after two or more solutions.
(default = 1)

0 Turns the combine solutions heuristic off.
1 Turns the combine solutions heuristic on.

coststrategy (string): how to use costs as priorities

This parameter influence the branching variable selection. If turned on, then the variables are sorted in order of their absolute costs, and branching is done first on variables with largest cost. This primitive strategy can be surprisingly effective.

(default = off)

off Turns off a specific cost strategy.
priorities Assigns highest priority to variables with largest absolute cost.
columnorder Assigns the priorities 1, 2, 3... with respect to the column ordering.
binaryfirst Handles two sets of priorities such that binary variables get high priority.
binarylast Handles two sets of priorities such that binary variables get low priority.
length Assigns high priority to variables that are at most nonzero.

crash (string): use crash method to get dual feasible

Determines whether CLP should use a crash algorithm to find a dual feasible basis.

(default = off)

off Switch off the creation of dual feasible basis by the crash method.
on Switch on the creation of dual feasible basis by the crash method.
solow halim Switch on a crash variant due to Solow and Halim.
halim solow Switch on a crash variant due to Solow and Halim with modifications of John J. Forrest.

crossover (integer): crossover to simplex algorithm after barrier

Determines whether CLP should crossover to the simplex algorithm after the barrier algorithm finished. Interior point algorithms do not obtain a basic solution. This option will crossover to a basic solution suitable for ranging or branch and cut.

(default = 1)

0 Turn off crossover to simplex algorithm after barrier algorithm finished.
1 Turn on crossover to simplex algorithm after barrier algorithm finished.

cutdepth (integer): depth in tree at which cuts are applied

If the depth in the tree is a multiple of cutdepth, then cut generators are applied. Cut generators may be off, on only at the root, on if they look useful, or on at some interval. Setting this option to a positive value K let CBC call a cutgenerator on a node whenever the depth in the tree is a multiple of K.

(default = -1)

-1 Does not turn on cut generators because the depth of the tree is a multiple of a value.

cutoff (real): cutoff for objective function value

CBC stops if the objective function values exceeds (in case of maximization) or falls below (in case of minimization) this value.

(default = GAMS cutoff)

cutoffconstraint (integer): whether to add a constraint from the objective function

For some problems, cut generators and general branching work better if the problem would be infeasible if the cost is too high. If this option is set, the objective function is added as a constraint which right hand side is set to the current cutoff value (objective value of best known solution).
(default = 0)

0 do not add extra constraint
1 add extra constraint

cuts (string): global switch for cutgenerators

A global switch to turn on or off the cutgenerators. This can be used to switch on or off all default cut generators. Then you can set individual ones off or on using the specific options.

(default = on)

off Turns off all cut generators.
on Turns on all default cut generators and CBC will try them in the branch and cut tree (see the option cutdepth on how to fine tune the behavior).
root Let CBC generate cuts only at the root node.
ifmove Let CBC use cut generators in the tree if they look as if they are doing some good and moving the objective value.
forceon Turns on all default cut generators and force CBC to use the cut generator at every node.

cut_passes_root (integer): number of cut passes at root node

Determines the number of rounds that the cut generators are applied in the root node. A negative value -n means that n passes are also applied if the objective does not drop. The default is to do 100 passes if the MIP has less than 500 columns, 100 passes (but stop if the drop in the objective function value is small) if it has less than 5000 columns, and 20 passes otherwise.
cut_passes_tree Determines the number of rounds that the cut generators are applied in the nodes of the tree other than the root node. A negative value -n means that n passes are also applied if the objective does not drop.

(default = 20 or 100)

cut_passes_slow (integer): number of cut passes for slow cut generators

Determines the number of rounds that slow cut generators should be applied. The idea is that the code does these cuts just a few times - less than the more usual cuts. The cut generators identified by "may be slow" at present are Lift and project cuts and both versions of Reduce and Split cuts.

(default = 10)

cut_passes_tree (integer): number of cut passes at nodes in the tree

(default = 1)

dins (integer): distance induced neighborhood search

This parameter control the use of the distance induced neighborhood search heuristic.

(default = 0)

0 Turns the distance induced neighborhood search off.
1 Turns the distance induced neighborhood search on.

divingcoefficient (integer): coefficient diving heuristic

This switches on the coefficient diving heuristic.

(default = 1)

0 Turns the coefficient diving heuristics off.
1 Turns the coefficient diving heuristics on.

divingfractional (integer): fractional diving heuristic

This switches on the fractional diving heuristic.

(default = 0)
0 Turns the fractional diving heuristics off.
1 Turns the fractional diving heuristics on.

**divingguided (integer):** guided diving heuristic
This switches on the guided diving heuristic.
(default = 0)
0 Turns the guided diving heuristics off.
1 Turns the guided diving heuristics on.

**divinglinesearch (integer):** line search diving heuristic
This switches on the line search diving heuristic.
(default = 0)
0 Turns the line search diving heuristics off.
1 Turns the linesearch diving heuristics on.

**divingpseudocost (integer):** pseudo cost diving heuristic
This switches on the pseudo costs diving heuristic.
(default = 0)
0 Turns the pseudo costs diving heuristics off.
1 Turns the pseudo costs diving heuristics on.

**divingrandom (integer):** turns on random diving heuristic
This switches on a random diving heuristic at various times.
(default = 0)
0 Turns the random diving heuristics off.
1 Turns the random diving heuristics on.

**divingvectorlength (integer):** vector length diving heuristic
This switches on the vector length diving heuristic.
(default = 0)
0 Turns the vector length diving heuristics off.
1 Turns the vector length diving heuristics on.

**dualpivot (string):** dual pivot choice algorithm
Choice of the pivoting strategy in the dual simplex algorithm.
(default = auto)
- auto Let CLP use a variant of the steepest choice method which starts like partial, i.e., scans only a subset of the primal infeasibilities, and later changes to full pricing when the factorization becomes denser.
- dantzig Let CLP use the pivoting strategy due to Dantzig.
- steepest Let CLP use the steepest choice method.
- partial Let CLP use a variant of the steepest choice method which scans only a subset of the primal infeasibilities to select the pivot step.

**dumpsolutions (string):** name of solutions index gdx file for writing alternate solutions
The name of a solutions index gdx file for writing alternate solutions found by CBC. The GDX file specified by this option will contain a set called index that contains the names of GDX files with the individual solutions.

**extravariables (integer):** group together variables with same cost
Switches on a trivial re-formulation that introduces extra integer variables to group together variables with same cost.
(default = 0)

**feaspump (integer):** feasibility pump

This parameter control the use of the feasibility pump heuristic at the root. This is due to Fischetti and Lodi and uses a sequence of LPs to try and get an integer feasible solution. Some fine tuning is available by the option **feaspump.passes**. Reference: M. Fischetti, F. Glover, and A. Lodi, The feasibility pump, Math. Programming, 104 (2005), pp. 91-104.
(default = 1)

- 0 Turns the feasibility pump off.
- 1 Turns the feasibility pump on.

**feaspump.passes (integer):** number of feasibility passes

This fine tunes the feasibility pump heuristic by setting the number of passes.
(default = 20)

**flowcovercuts (string):** Flow Cover Cuts

Determines whether and when CBC should try to generate flow cover cuts. See the option **cuts** for an explanation on the different values. The flow cover cut generator generates lifted simple generalized flow cover inequalities. Since flow cover inequalities are generally not facet-defining, they are lifted to obtain stronger inequalities. Although flow cover inequalities requires a special problem structure to be generated, they are quite useful for solving general mixed integer linear programs. Reference: Z. Gu, G.L. Nemhauser, M.W.P. Savelsbergh, Lifted flow cover inequalities for mixed 0-1 integer programs, Math. Programming A 85 (1999) 439-467.
(default = ifmove)

**gomorycuts (string):** Gomory Cuts

(default = ifmove)

**gomorycuts2 (string):** Gomory Cuts 2nd implementation

(default = off)

- off Turns off the cut generators.
- on Turns on the cut generator and CBC will try it in the branch and cut tree (see the option **cutdepth** on how to fine tune the behavior).
- root Let CBC generate gomory cuts only at the root node.
- ifmove Let CBC use this cut generator in the tree if it looks as if it is doing some good and moves the objective value.
- forceon Turns on the cut generator and forces CBC to use it at every node.
- longroot
- endonly
- long
- longifmove
forcelongon
longendonly

greedyheuristic (string): greedy heuristic

This parameter controls the use of a pair of greedy heuristics which will try to obtain a solution. It may just fix a percentage of variables and then try a small branch and cut run.

(default = on)

off Turns off the greedy heuristic.
on Turns on the greedy heuristic.
root Turns on the greedy heuristic only for the root node.

heuristics (integer): global switch for heuristics

This parameter can be used to switch on or off all heuristics, except for the local tree search as it dramatically alters the search. Then you can set individual ones off or on.

(default = 1)

0 Turns all MIP heuristics off.
1 Turns all MIP heuristics on (except local tree search).

idiotcrash (integer): idiot crash

This is a type of ‘crash’ which works well on some homogeneous problems. It works best on problems with unit elements and right hand sides but will do something to any model. It should only be used before the primal simplex algorithm. A positive number determines the number of passes that idiotcrash is called.

(default = -1)

-1 Let CLP decide by itself whether to use it.
0 Switch this method off.

increment (real): increment of cutoff when new incumbent

A valid solution must be at least this much better than last integer solution. If this option is not set then it CBC will try and work one out. E.g., if all objective coefficients are multiples of 0.01 and only integer variables have entries in objective then this can be set to 0.01.

(default = GAMS cheat)

iterlim (integer): iteration limit

For an LP, this is the maximum number of iterations to solve the LP. For a MIP, this option is ignored.

(default = GAMS iterlim)

knapsackcuts (string): Knapsack Cover Cuts

Determines whether and when CBC should try to generate knapsack cover cuts. See the option cuts for an explanation on the different values. The knapsack cover cut generator looks for a series of different types of minimal covers. If a minimal cover is found, it lifts the associated minimal cover inequality and adds the lifted cut to the cut set. Reference: S. Martello, and P. Toth, Knapsack Problems, Wiley, 1990, p30.

(default = ifmove)

liftandprojectcuts (string): Lift and Project Cuts

Determines whether and when CBC should try to generate lift and project cuts. They might be expensive to compute, thus they are switched off by default. See the option cuts for an explanation on the different values. Reference: E. Balas and M. Perregaard, A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. Math. Program., 94(203,Ser. B):221-245,2003.

(default = off)
**localtreesearch (integer):** local tree search heuristic

This parameter control the use of a local search algorithm when a solution is found. It is from Fischetti and Lodi and is not really a heuristic although it can be used as one (with limited functionality). This heuristic is not controlled by the option `heuristics`. Reference: M. Fischetti and A. Lodi, Local Branching, Math. Programming B, 98 (2003), pp. 23-47.

(default = 0)

0 Turns the local tree search off.

1 Turns the local tree search on.

**loglevel (integer):** CBC loglevel

Amount of output to print by CBC.

(default = 1)

**maxfactor (integer):** maximum number of iterations between refactorizations

Maximum number of iterations between refactorizations in CLP. If this is left at the default value of 200 then CLP will guess at a value to use. CLP may decide to refactorize earlier for accuracy.

(default = 200)

**maxsol (integer):** maximal number of solutions to store during search

Maximal number of solutions to store during search and to dump into gdx files if dumpsolutions options is set.

(default = 100)

**mipstart (integer):** whether it should be tried to use the initial variable levels as initial MIP solution

This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the variable level values should be checked to see if they provide an integer feasible solution before starting optimization.

(default = 0)

0 Do not use the initial variable levels.

1 Try to use the initial variable levels as a MIP starting solution.

**mircuts (string):** Mixed Integer Rounding Cuts

Determines whether and when CBC should try to generate mixed integer rounding cuts. See the option `cuts` for an explanation on the different values. Reference: H. Marchand and L. A. Wolsey, Aggregation and Mixed Integer Rounding to Solve MIPs, Operations Research, 49(3), (2001).

(default = ifmove)

**multiplerootpasses (integer):** runs multiple copies of the solver at the root node

Solves the root node with multiple copies of the solver, each with its own different seed and collects the solutions and cuts so that the main solver has a richer set of solutions and possibly stronger cuts. If threads are enabled, then this is done in parallel (Attention: doing this in parallel is experimental). If the number is below 100, then the option gives the number of times the root is solved. The actual format is aabbcc where aa is number of extra passes, if bb is non zero then it is number of threads to use (otherwise uses threads setting) and cc is number of times to do root phase. The solvers do not interact with each other. However if extra passes are specified then cuts are collected and used in later passes - so there is interaction there.

(default = 0)

**naiveheuristics (integer):** naive heuristics

This parameter controls the use of some naive heuristics, e.g., fixing of all integers with costs to zero.

(default = 0)

0 Turns the naive heuristics off.

1 Turns the naive heuristics on.
nodelim (integer): node limit

Maximum number of nodes that are considered in the Branch and Bound.
(default = GAMS nodlim)

nodestrategy (string): how to select nodes

This determines the strategy used to select the next node from the branch and cut tree.
(default = fewest)

hybrid Let CBC do first a breath search on nodes with a small depth in the tree and then switch to choose nodes with fewest infeasibilities.
fewest This will let CBC choose the node with the fewest infeasibilities.
depth This will let CBC always choose the node deepest in tree. It gives minimum tree size but may take a long time to find the best solution.
upfewest This will let CBC choose the node with the fewest infeasibilities and do up branches first.
downfewest This will let CBC choose the node with the fewest infeasibilities and do down branches first.
updepth This will let CBC choose the node deepest in tree and do up branches first.
downdepth This will let CBC choose the node deepest in tree and do down branches first.

nodelim (integer): node limit

Maximum number of nodes that are considered in the Branch and Bound. This option is overwritten by nodelim, if set.
(default = GAMS nodlim)

optca (real): absolute stopping tolerance

Absolute optimality criterion for a MIP. CBC stops if the gap between the best known solution and the best possible solution is less than this value.
(default = GAMS optca)

optcr (real): relative stopping tolerance

Relative optimality criterion for a MIP. CBC stops if the relative gap between the best known solution and the best possible solution is less than this value.
(default = GAMS optcr)

parallelmode (string): whether to run opportunistic or deterministic

Determines whether a parallel MIP search (threads > 1) should be done in a deterministic (i.e., reproducible) way or in a possibly faster but not necessarily reproducible way.
(default = deterministic)

opportunistic
deterministic

passpresolve (integer): how many passes to do in presolve

Normally Presolve does 5 passes but you may want to do less to make it more lightweight or do more if improvements are still being made. As Presolve will return if nothing is being taken out, you should not normally need to use this fine tuning.
(default = 5)

perturbation (integer): perturbation of problem

Determines whether CLP should perturb the problem before starting. Perturbation helps to stop cycling, but CLP uses other measures for this. However, large problems and especially ones with unit elements and unit right hand sides or costs benefit from perturbation. Normally CLP tries to be intelligent, but you can switch this off.
(default = 1)

0 Turns off perturbation of LP.
1 Turns on perturbation of LP.

**pivotandfix (integer): pivot and fix heuristic**

This parameter controls the use of the pivot and fix heuristic.

(default = 0)

0 Turns the naive pivot and fix heuristic off.
1 Turns the naive pivot and fix heuristic on.

**preprocess (string): integer presolve**

This option controls the MIP specific presolve routines. They try to reduce the size of the model in a similar way to presolve and also try to strengthen the model. This can be very useful and is worth trying.

(default = on)

off Turns off the presolve routines.
on Turns on the presolve routines.
equa1 Turns on the presolve routines and let CBC turn inequalities with more than 5 elements into equalities (cliques) by adding slack variables.
equa1a11 Turns on the presolve routines and let CBC turn all inequalities into equalities by adding slack variables.
sos This option let CBC search for rows with upper bound 1 and where all nonzero coefficients are 1 and creates special ordered sets if the sets are not overlapping and all integer variables (except for at most one) are in the sets.
try sos This option is similar to sos, but allows any number integer variables to be outside of the sets.

**presolve (integer): switch for initial presolve of LP**

Presolve analyzes the model to find such things as redundant constraints, constraints which fix some variables, constraints which can be transformed into bounds, etc. For the initial solve of any problem this is worth doing unless you know that it will have no effect.

(default = 1)

0 Turns off the initial presolve.
1 Turns on the initial presolve.

**primalpivot (string): primal pivot choice algorithm**

Choice of the pivoting strategy in the primal simplex algorithm.

(default = auto)

auto Let CLP use a variant of the exact devex method.
dantzig Let CLP use the pivoting strategy due to Dantzig.
steepest Let CLP use the steepest choice method.
partial Let CLP use a variant of the exact devex method which scans only a subset of the primal infeasibilities to select the pivot step.
exact Let CLP use the exact devex method.
change Let CLP initially use Dantzig pivot method until the factorization becomes denser.

**printfrequency (integer): frequency of status prints**

Controls the number of nodes that are evaluated between status prints.
Automatic choice, which is 100 for large problems and 1000 for small problems.

**probingcuts (string): Probing Cuts**

Determines whether and when CBC should try to generate cuts based on probing. Additional to the values for the option cuts three more values are possible here. Reference: M. Savelsbergh, Preprocessing and Probing Techniques for Mixed Integer Programming Problems, ORSA Journal on Computing 6 (1994), 445.

(default = ifmove)

- off Turns off Probing.
- on Turns on Probing and CBC will try it in the branch and cut tree (see the option cutdepth how to fine tune this behavior).
- root Let CBC do Probing only at the root node.
- ifmove Let CBC do Probing in the tree if it looks as if it is doing some good and moves the objective value.
- forceon Turns on Probing and forces CBC to do Probing at every node.
- forceonbut Turns on Probing and forces CBC to call the cut generator at every node, but does only probing, not strengthening etc.
- forceonstrong If CBC is forced to turn Probing on at every node (by setting this option to force), but this generator produces no cuts, then it is actually turned on only weakly (i.e., just every now and then). Setting forceonstrong forces CBC strongly to do probing at every node.
- forceonbutstrong This is like forceonstrong, but does only probing (column fixing) and turns off row strengthening, so the matrix will not change inside the branch and bound.

**proximitysearch (integer): proximity search heuristic**

This parameter controls the use of the "No-Neighborhood Search" 0-1 MIP refinement heuristic proposed by Fischetti and Monaci (2012). The idea is to define a sub-MIP without additional constraints but with a modified objective function intended to attract the search in the proximity of the incumbent. The approach works well for 0-1 MIPs whose solution landscape is not too irregular (meaning the there is reasonable probability of finding an improved solution by flipping a small number of binary variables), in particular when it is applied to the first heuristic solutions found at the root node.

(default = 0)

- 0 Turns the proximity heuristic off.
- 1 Turns the proximity heuristic on.

**randomizedrounding (integer): randomized rounding heuristic**

This parameter controls the use of the randomized rounding heuristic.

(default = 0)

- 0 Turns the randomized rounding heuristic off.
- 1 Turns the randomized rounding heuristic on.

**randomseedcbc (integer): random seed for CBC**

Allows initialization of the random seed for pseudo-random numbers used in heuristics such as the Feasibility Pump to decide whether to round up or down. The special value of 0 lets Cbc use the time of the day for the initial seed.

(default = -1)

**randomseedclp (integer): random seed for CLP**

Allows initialization of the random seed for pseudo-random numbers used to break ties in degenerate problems. This may yield a different continuous optimum and, in the context of Cbc, different cuts and heuristic solutions. The special value of 0 lets Clp use the time of the day for the initial seed.
reduceandsplitcuts (string): Reduce and Split Cuts

Determines whether and when CBC should try to generate reduced and split cuts. See the option cuts for an explanation on the different values. Reduce and split cuts are variants of Gomory cuts. Starting from the current optimal tableau, linear combinations of the rows of the current optimal simplex tableau are used for generating Gomory cuts. The choice of the linear combinations is driven by the objective of reducing the coefficients of the non basic continuous variables in the resulting row. Reference: K. Anderson, G. Cornuejols, and Yanjun Li, Reduce-and-Split Cuts: Improving the Performance of Mixed Integer Gomory Cuts, Management Science 51 (2005).

(default = off)

reduceandsplitcuts2 (string): Reduce and Split Cuts 2nd implementation

Determines whether and when CBC should try to generate reduced and split cuts using the 2nd implementation. Reduce and split cuts are variants of Gomory cuts. Reference: G. Cornuejols and G. Nannicini, Practical strategies for generating rank-1 split cuts in mixed-integer linear programming, Mathematical Programming Computation 3 (2011), 281-318.

(default = off)

  off Turns off all cut generators.
  on Turns on all default cut generators and CBC will try them in the branch and cut tree (see the option cutdepth on how to fine tune the behavior).
  root Let CBC generate cuts only at the root node.
  longon
  longroot

rens (integer): relaxation enforced neighborhood search

This parameter controls the use of the relaxation enforced neighborhood search heuristic.

(default = 0)

  0 Turns the relaxation enforced neighborhood search off.
  1 Turns the relaxation enforced neighborhood search on.

residualcapacitycuts (string): Residual Capacity Cuts


(default = off)

reslim (real): resource limit

Maximum time in seconds.

(default = GAMS reslim)

rins (integer): relaxed induced neighborhood search

This parameter controls the use of the relaxed induced neighborhood search heuristic. This heuristic compares the current solution with the best incumbent, fixes all discrete variables with the same value, presolves the problem, and does a branch and bound for 200 nodes. Reference: E. Danna, E. Rothberg, and C. Le Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, Math. Programming, 102 (1) (2005), pp. 71-91.

(default = 0)
0 Turns the relaxed induced neighborhood search off.
1 Turns the relaxed induced neighborhood search on.

**Roundingheuristic (integer): rounding heuristic**

This parameter control the use of a simple (but effective) rounding heuristic at each node of tree.

(default = 1)

0 Turns the rounding heuristic off.
1 Turns the rounding heuristic on.

**Scaling (string): scaling method**

Scaling can help in solving problems which might otherwise fail because of lack of accuracy. It can also reduce the number of iterations. It is not applied if the range of elements is small. Both methods do several passes alternating between rows and columns using current scale factors from one and applying them to the other.

(default = auto)

off Turns off scaling.

auto Let CLP choose the scaling method automatically. It decides for one of these methods depending on which gives the better ratio of the largest element to the smallest one.

equilibrium Let CLP use an equilibrium based scaling method which uses the largest scaled element.

geometric Let CLP use a geometric based scaling method which uses the squareroot of the product of largest and smallest element.

**Sifting (integer): synonym for sprint crash**

Synonym for sprintcrash.

(default = -1)

**Sollim (integer): limit on number of solutions**

A limit on number of feasible solutions that CBC should find for a MIP.

(default = -1)

-1 No limit on the number of feasible solutions.

**Solvefinal (integer): final solve of MIP with fixed discrete variables**

whether the MIP with discrete variables fixed to solution values should be solved after CBC finished

(default = 1)

0 Turn off the final LP solve.
1 Turn on the final LP solve.

**Solvetrace (string): name of trace file for solving information**

Name of file for writing solving progress information during solve.

**Solvetracenodefreq (integer): frequency in number of nodes for writing to solve trace file**

frequency in number of nodes for writing solving progress information

(default = 100)

**Solvetracetimefreq (real): frequency in seconds for writing to solve trace file**

frequency in seconds for writing solving progress information

(default = 5)

**Special (string): options passed unseen to CBC**

This parameter let you specify CBC options which are not supported by the GAMS/CBC interface. The string
value given to this parameter is split up into parts at each space and added to the array of parameters given to CBC (in front of the -solve command). Hence, you can use it like the command line parameters for the CBC standalone version.

sprintcrash\hspace{1em} (integer): sprint crash

For long and thin problems this method may solve a series of small problems created by taking a subset of the columns. Cplex calls it ‘sifting’. A positive number determines the number of passes that sprintcrash is called.

(dedault = -1)
-1 Let CLP decide by itself whether to use it.
0 Switch this method off.

startalg\hspace{1em} (string): LP solver for root node

Determines the algorithm to use for an LP or the initial LP relaxation if the problem is a MIP.

(dedault = dual)

primal\hspace{1em} Let CLP use the primal simplex algorithm.
dual\hspace{1em} Let CLP use the dual simplex algorithm.
barrier\hspace{1em} Let CLP use a primal dual predictor corrector algorithm.

strategy\hspace{1em} (integer): switches on groups of features

Setting strategy to 1 (the default) uses Gomory cuts using tolerance of 0.01 at root, does a possible restart after 100 nodes if Cbc can fix many variables and activates a diving and RINS heuristic and makes feasibility pump more aggressive.

(dedault = 1)

0 Use this setting for easy problems.
1 This is the default setting.
2 Use this setting for difficult problems.

strongbranching\hspace{1em} (integer): strong branching

Determines the number of variables to look at in strong branching. In order to decide which variable to branch on, the code will choose up to this number of unsatisfied variables and try minimal up and down branches. The most effective one is chosen. If a variable is branched on many times then the previous average up and down costs may be used - see the option trustpseudocosts.

(dedault = 5)

threads\hspace{1em} (integer): number of threads to use

This option controls the multithreading feature of CBC. A number between 1 and 99 sets the number of threads used for parallel branch and bound.

(dedault = GAMS\hspace{1em} threads)

tol_dual\hspace{1em} (real): dual feasibility tolerance

The maximum amount the dual constraints can be violated and still be considered feasible.

(dedault = 1e-7)

tol_integer\hspace{1em} (real): tolerance for integrality

For an optimal solution, no integer variable may be farther than this from an integer value.

(dedault = 1e-6)

tol_presolve\hspace{1em} (real): tolerance used in presolve

The tolerance used in presolve.

(dedault = 1e-8)
**tol_primal** *(real):* primal feasibility tolerance

The maximum amount the primal constraints can be violated and still be considered feasible.

(default = 1e-7)

**trustpseudocosts** *(integer):* after howmany nodes we trust the pseudo costs

Using strong branching computes pseudo-costs. This parameter determines after how many branches for a variable we just trust the pseudo costs and do not do any more strong branching.

(default = 5)

**twomircuts** *(string):* Two Phase Mixed Integer Rounding Cuts

Determines whether and when CBC should try to generate *two phase mixed integer rounding cuts*. See the option cuts for an explanation on the different values. Reference: S. Dash, and O. Guenluke, Valid Inequalities Based on Simple Mixed-integer Sets, to appear in Math. Programming.

(default = root)

**vubheuristic** *(integer):* VUB heuristic

This parameter control the use of the VUB heuristic. If it is set (between -2 and 20), Cbc will try and fix some integer variables.

**writemps** *(string):* create MPS file for problem

Write the problem formulation in MPS format. The parameter value is the name of the MPS file.

**zerohalfcuts** *(string):* Zero-Half Cuts

Determines whether and when CBC should try to generate zero-half cuts. So far, they may help only on a small subset of problems and may need some tuning. Reference: G. Andreello, A. Caprara, and M. Fischetti, Embedding Cuts in a Branch and Cut Framework: a Computational Study with \(\{0,1/2\}\)-Cuts, INFORMS Journal on Computing 19 (2007), 229-238.

(default = off)

- **off** Turns off the zero half cuts generators.
- **on** Turns on the zero half cuts cut generators and CBC will try it in the branch and cut tree (see the option cutdepth on how to fine tune the behavior).
- **root** Let CBC generate zero half cuts only at the root node.
- **ifmove** Let CBC use the zero half cuts generator in the tree if it looks as if it is doing some good and moves the objective value.
- **forceon** Turns on the zero half cuts generator and forces CBC to use it at every node.
- **onglobal**
CONVERT

Contents

1 Introduction .......................................................... 649
2 How to use CONVERT .................................................. 649
3 The GAMS Scalar Format ............................................. 649
4 User-Specified Options .............................................. 653
   4.1 Target languages ................................................. 653
   4.2 Other options ...................................................... 654

1 Introduction

CONVERT is a utility which transforms a GAMS model instance into a scalar model where all confidential information has been removed or into formats used by other modeling and solution systems. CONVERT is designed to achieve the following goals:

- Permit users to convert a confidential model into GAMS scalar format so that any identifiable structure is removed. It can then be passed on to others for investigation without confidentiality being lost.
- A way of sharing GAMS test problems for use with other modeling systems or solvers.

CONVERT comes free of charge with any licensed GAMS system and can convert GAMS models into a number of formats, see Section Target languages for a list.

2 How to use CONVERT

CONVERT is run like any other GAMS solver. From the command line this is:

```plaintext
>> gams modelname modeltype=convert
```

where `modelname` is the GAMS model name and `modeltype` the solver indicator for a particular model type (e.g. LP, MIP, RMIP, QCP, MIQCP, RMIQCP, NLP, DNLP, CNS, MINLP, or MCP). CONVERT can also be specified via the option statement within the model itself before the solve statement:

```plaintext
option modeltype=convert;
```

3 The GAMS Scalar Format

By default, CONVERT generates a scalar GAMS model (`gams.gms`) from the input model. The scalar model exhibits the following characteristics:
• A model without sets or indexed parameters. It does not exhibit any of the advanced characteristics of modeling systems and is easily transformable.

• A model with a new set of individual variables, depicting each variable in the GAMS model as one of 3 types: positive, integer or binary. Each variable is numbered sequentially, i.e. all positive GAMS variables are mapped into n single variables x1, x2, ..., xn.

• A model with individual equations depicting each variable in the GAMS model. All equations are also numbered sequentially, that is equations e1, e2, ..., en.

Equation and variable bounds, as well as variable starting values are preserved from the original GAMS formulation.

As an example, suppose the user wishes to translate the GAMS Model Library model transport into scalar format, One would run

gams transport.gms lp=convert

which would generate the following scalar model gams.gms:

* LP written by GAMS Convert at 07/29/04 12:59:58
*
* Equation counts
* Total   E   G   L   N   X   C
*       6   1   3   2   0   0   0
*
* Variable counts
* x   b   i   s1s   s2s   sc   si
* Total   cont   binary   integer   sos1   sos2   scont   sint
*       7       7       0       0       0       0       0       0
* FX      0       0       0       0       0       0       0       0
*
* Nonzero counts
* Total   const   NL   DLL
*       19       19       0       0
*
* Solve m using LP minimizing x7;

Variables  x1,x2,x3,x4,x5,x6,x7;
Positive Variables  x1,x2,x3,x4,x5,x6;
Equations  e1,e2,e3,e4,e5,e6;

e1..  - 0.225*x1 - 0.153*x2 - 0.162*x3 - 0.225*x4 - 0.162*x5 - 0.126*x6 + x7 =E=  0;
e2..  x1 + x2 + x3 =L=  350;
e3..  x4 + x5 + x6 =L=  600;
e4..  x1 + x4 =Q=  325;
e5..  x2 + x5 =Q=  300;
e6..  x3 + x6 =Q=  275;

* set non default bounds

* set non default levels

* set non default marginals

Model m / all /;
m.limrow=0; m.limcol=0;
Solve m using LP minimizing x7;

Note that the resulting scalar model does not contain any of the descriptive information about the data or the context of the constraints.

Additionally, a dictionary file (dict.txt) is created by default which specifies a mapping between the variable and equation names in the scalar model and their corresponding names in the original model.

For the above example, the dictionary file is

LP written by GAMS Convert at 07/29/04 12:59:59

Equation counts

<table>
<thead>
<tr>
<th>Total</th>
<th>E</th>
<th>G</th>
<th>L</th>
<th>N</th>
<th>X</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Variable counts

<table>
<thead>
<tr>
<th>x</th>
<th>b</th>
<th>i</th>
<th>s1s</th>
<th>s2s</th>
<th>sc</th>
<th>si</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Nonzero counts

<table>
<thead>
<tr>
<th>Total</th>
<th>const</th>
<th>NL</th>
<th>DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Equations 1 to 6

e1  cost

e2  supply(seattle)
e3  supply(san-diego)
e4  demand(new-york)
e5  demand(chicago)
e6  demand(topeka)

Variables 1 to 7

x1  x(seattle,new-york)
x2  x(seattle,chicago)
x3  x(seattle,topeka)
x4  x(san-diego,new-york)
x5  x(san-diego,chicago)
x6  x(san-diego,topeka)
x7  z

Conversion of a GAMS model to a scalar one may be handy for model debugging. However, in this case, it may be good to retain the original variable and equation names. The following simple sed command attempts to achieve this:

```
sed -n -e "s:^ *[exbi][0-9]*:s/^\([\d]*\):s/\1/\2:g:gp" dict.txt | sed -n '1!G;h;$p' > mod.txt
```

For the above example, this outputs:

```
Variables x(seattle,new-york),x(seattle,chicago),x(seattle,topeka),x(san-diego,new-york),
x(san-diego,chicago),x(san-diego,topeka),x(san-diego,new-york),x(san-diego,chicago),x(san-diego,topeka);
```

Positive Variables x(seattle,new-york),x(seattle,chicago),x(seattle,topeka),
x(san-diego,new-york),x(san-diego,chicago),x(san-diego,topeka);
Equations cost, supply(seattle), supply(san-diego), demand(new-york), demand(chicago), demand(topeka);

cost.. - 0.225*x(seattle,new-york) - 0.153*x(seattle,chicago) - 0.162*x(seattle,topeka) - 0.225*x(san-diego,new-york) - 0.162*x(san-diego,chicago) - 0.126*x(san-diego,topeka) + z =E= 0;
supply(seattle).. x(seattle,new-york) + x(seattle,chicago) + x(seattle,topeka) =L= 350;
supply(san-diego).. x(san-diego,new-york) + x(san-diego,chicago) + x(san-diego,topeka) =L= 600;
demand(new-york).. x(seattle,new-york) + x(san-diego,new-york) =G= 325;
demand(chicago).. x(seattle,chicago) + x(san-diego,chicago) =G= 300;
demand(topeka).. x(seattle,topeka) + x(san-diego,topeka) =G= 275;

Of course, this is not a valid GAMS code and cannot be compiled, but it may be sufficient to view the model algebra as generated by the GAMS compiler.

By using

```
sed -n -e "y/(),-/____/" -e "s:^ \([\text{exbi}]\[0-9]\) \(.*\) :s/\1/\2/g:gp" dict.txt | sed -n '1!G;h;$p' > mod.txt
```

```
sed -f mod.txt gams.gms
```

one gets for this example

```
Variables x_seattle_new_york_, x_seattle_chicago_, x_seattle_topeka_, x_san_diego_new_york_, x_san_diego_chicago_, x_san_diego_topeka_, z;
Positive Variables x_seattle_new_york_, x_seattle_chicago_, x_seattle_topeka_, x_san_diego_new_york_, x_san_diego_chicago_, x_san_diego_topeka_;

Equations cost, supply_seattle_, supply_san_diego_, demand_new_york_, demand_chicago_, demand_topeka_;

cost.. - 0.225*x_seattle_new_york_ - 0.153*x_seattle_chicago_ - 0.162*x_seattle_topeka_ - 0.225*x_san_diego_new_york_ - 0.162*x_san_diego_chicago_ - 0.126*x_san_diego_topeka_ + z =E= 0;
supply_seattle_.. x_seattle_new_york_ + x_seattle_chicago_ + x_seattle_topeka_ =L= 350;
supply_san_diego_.. x_san_diego_new_york_ + x_san_diego_chicago_ + x_san_diego_topeka_ =L= 600;
demand_new_york_.. x_seattle_new_york_ + x_san_diego_new_york_ =G= 325;
demand_chicago_.. x_seattle_chicago_ + x_san_diego_chicago_ =G= 300;
demand_topeka_.. x_seattle_topeka_ + x_san_diego_topeka_ =G= 275;
```

This can even be compiled by GAMS and gives the correct solution.

The proposed commands come with several limitations and may not produce in all cases the desired output. For example, wrong results would be printed if the original model contains variable or equation names that start with `{b,i,e,x}`[digit]. Also semicontinuous or semiinteger variables or special ordered sets are not supported by the above. We leave it to the experienced user to extend the command appropriately.
4 User-Specified Options

CONVERT options are passed on through option files. If you specify `<modelname>.optfile = 1;` before the SOLVE statement in your GAMS model, CONVERT will look for and read an option file with the name `convert.opt` (see The Solver Option File for general use of solver option files). The syntax for the CONVERT option file is

```
optname value
```

with one option on each line. For example,

```
ampl
```

This option file would tell CONVERT to produce an AMPL input file. For file format options, the user can specify the filename for the file to be generated. For example, the option file entry

```
lingo myfile.lng
```

would generate a LINGO input file format called `myfile.lng`. Using the option `lingo` by itself, would produce the default output file for that option (`lingo.lng`).

All available options are listed in the following tables.

### 4.1 Target languages

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Generates all supported file formats</td>
<td></td>
</tr>
<tr>
<td>AlphaECP</td>
<td>Generates AlphaECP input file</td>
<td>alpha.ecp</td>
</tr>
<tr>
<td>Ampl</td>
<td>Generates Ampl input file</td>
<td>ampl.mod</td>
</tr>
<tr>
<td>AmplNLC</td>
<td>Generate Ampl NLC compatible file</td>
<td>amplnlc.c</td>
</tr>
<tr>
<td>Analyze</td>
<td>Generates three text files for rows columns and matrix</td>
<td>analyze.txt</td>
</tr>
<tr>
<td>AnalyzeS</td>
<td>Generates short form of Analyze</td>
<td>analyzes.txt</td>
</tr>
<tr>
<td>Baron</td>
<td>Generates Baron input file</td>
<td>gams.bar</td>
</tr>
<tr>
<td>CplexLP</td>
<td>Generate CPLEX LP format input file</td>
<td>cplex.lp</td>
</tr>
<tr>
<td>CplexMPS</td>
<td>Generate CPLEX MPS format input file</td>
<td>cplex.mps</td>
</tr>
<tr>
<td>Ddict</td>
<td>Generate Convert to GAMS Dictionary Map</td>
<td>dictmap.gdx</td>
</tr>
<tr>
<td>FileList</td>
<td>Generate file list of file formats generated</td>
<td>files.txt</td>
</tr>
<tr>
<td>FixedMPS</td>
<td>Generate fixed format MPS file</td>
<td>fixed.mps</td>
</tr>
<tr>
<td>Gams</td>
<td>Generate GAMS scalar model. This is the default conversion format used.</td>
<td>gams.gms</td>
</tr>
<tr>
<td>Lgo</td>
<td>Generate an LGO Fortran file</td>
<td>lgomain.for</td>
</tr>
<tr>
<td>LindoMPI</td>
<td>Generate Lindo MPI file</td>
<td>lindo.mpi</td>
</tr>
<tr>
<td>Lingo</td>
<td>Generate Lingo input file</td>
<td>lingo.lng</td>
</tr>
<tr>
<td>LocalSolver</td>
<td>Generate LocalSolver input file (only with ConvertD)</td>
<td>localsolver.lsp</td>
</tr>
<tr>
<td>LSPSol</td>
<td>Generate Output function in LocalSolver input file (only with ConvertD)</td>
<td>lpsol.gms</td>
</tr>
<tr>
<td>Memo</td>
<td>Generate a memo file containing model statistics and files created.</td>
<td>memo.text</td>
</tr>
<tr>
<td>Minopt</td>
<td>Generate Minopt input file</td>
<td>minopt.dat</td>
</tr>
<tr>
<td>NLP2MCP</td>
<td>Generates GAMS scalar MCP model</td>
<td>gamsmpc.gms</td>
</tr>
<tr>
<td>OSiL</td>
<td>Generates Optimization Services instance Language (OSiL) file</td>
<td>osil.xml</td>
</tr>
<tr>
<td>Pyomo</td>
<td>Generates Pyomo Concrete scalar model</td>
<td>gams.py</td>
</tr>
</tbody>
</table>
### 4.2 Other options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConeReform</td>
<td>Reformulation of cone =C= constraints to NLP format</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 keep =C= format</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 convert conic constraints to NLP format</td>
<td></td>
</tr>
<tr>
<td>GmsInsert</td>
<td>Line to be inserted before the solve statement</td>
<td>$if NOT 'gams.u1' == '' $include 'gams.u1'</td>
</tr>
<tr>
<td>headerTimeStamp</td>
<td>Control format of time stamp in header of output file</td>
<td>default</td>
</tr>
<tr>
<td></td>
<td>default Use the traditional default timestamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none Use no timestamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>svnId Use the SVN keyword Id as a timestamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>svnLastChangedDate Use the SVN keyword LastChangedDate as a timestamp</td>
<td></td>
</tr>
<tr>
<td>Hessian</td>
<td>Writes GDX version of current point - Jacobian and Hessian</td>
<td>hessian.gdx</td>
</tr>
<tr>
<td>Include</td>
<td>Start reading from a new file</td>
<td></td>
</tr>
<tr>
<td>intervalEval</td>
<td>Include interval evaluations in Jacobian and Hessian</td>
<td>0</td>
</tr>
<tr>
<td>intervalEvalDebug</td>
<td>Turn on debug output for the interval evaluations</td>
<td>0</td>
</tr>
<tr>
<td>Jacobian</td>
<td>Writes GDX version of current point - Jacobian</td>
<td>jacobian.gdx</td>
</tr>
<tr>
<td>Match</td>
<td>Force a complete match for all MCP variable/equation pairs</td>
<td></td>
</tr>
<tr>
<td>ObjVar</td>
<td>Name of objective variable</td>
<td>GAMS index name, e.g. x1</td>
</tr>
<tr>
<td>Reform</td>
<td>Force reformulations</td>
<td>100</td>
</tr>
<tr>
<td>Terminate</td>
<td>Force GAMS to terminate after conversion</td>
<td></td>
</tr>
</tbody>
</table>
**Couenne**

**Contents**

1 Usage .................................................................................................................. 656  
1.1 Specification of Options .................................................................................... 656  
2 List of Options ...................................................................................................... 656  
  2.1 Couenne options ............................................................................................ 656  
  2.2 Bonmin Branch-and-bound options ............................................................... 661  
  2.3 Bonmin NLP interface .................................................................................. 661  
  2.4 Bonmin NLP solution robustness .................................................................. 661  
  2.5 Bonmin Nonconvex problems ....................................................................... 662  
  2.6 Bonmin Outer Approximation cuts generation ............................................. 662  
  2.7 Bonmin Output and Loglevel ......................................................................... 662  
  2.8 Bonmin Strong branching setup ..................................................................... 663  
  2.9 Ipopt Barrier Parameter Update ................................................................. 663  
  2.10 Ipopt Convergence ..................................................................................... 664  
  2.11 Ipopt Hessian Approximation ..................................................................... 665  
  2.12 Ipopt Initialization ...................................................................................... 665  
  2.13 Ipopt Line Search ....................................................................................... 665  
  2.14 Ipopt Linear Solver .................................................................................... 666  
  2.15 Ipopt MA27 Linear Solver ........................................................................... 667  
  2.16 Ipopt MA28 Linear Solver ........................................................................... 667  
  2.17 Ipopt MA57 Linear Solver ........................................................................... 667  
  2.18 Ipopt MA77 Linear Solver ........................................................................... 667  
  2.19 Ipopt MA86 Linear Solver ........................................................................... 668  
  2.20 Ipopt MA97 Linear Solver ........................................................................... 668  
  2.21 Ipopt Mumps Linear Solver ......................................................................... 668  
  2.22 Ipopt NLP .................................................................................................... 669  
  2.23 Ipopt NLP Scaling ....................................................................................... 669  
  2.24 Ipopt Output ................................................................................................ 669  
  2.25 Ipopt Pardiso Linear Solver ......................................................................... 670  
  2.26 Ipopt Restoration Phase .............................................................................. 670  
  2.27 Ipopt Step Calculation ................................................................................ 670  
  2.28 Ipopt Warm Start ........................................................................................ 671  
3 Detailed Options Description ................................................................................ 671  
Bibliography ............................................................................................................. 718

COIN-OR Couenne (Convex Over and Under Envelopes for Nonlinear Estimation) is an open-source solver for nonconvex mixed-integer nonlinear programming (MINLPs). The code has been developed originally in a cooperation of Carnegie Mellon University and IBM Research. The COIN-OR project leader for Couenne is Pietro Belotti, now with FICO, Ltd.

Couenne solves convex and nonconvex MINLPs by an LP based spatial branch-and-bound algorithm. The implementation extends BONMIN by routines to compute valid linear outer approximations for nonconvex problems and methods for bound tightening and branching on nonlinear variables. Couenne uses IPOPT to solve NLP subproblems.
For more information on the algorithm we refer to [2, 3] and the Couenne web site. Most of the Couenne documentation in this section is taken from the Couenne manual [1].

Couenne can handle mixed-integer nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. Further, an algebraic description of the model need to be made available, which makes the use of some GAMS functions and user-specified external functions impossible. The Couenne link in GAMS supports continuous, binary, and integer variables, but no special ordered sets, semi-continuous or semi-integer variables (see chapter 17.1 of the GAMS User’s Guide).

1 Usage

The following statement can be used inside your GAMS program to specify using Couenne

Option MINLP = COUENNE; { or LP, RMIP, MIP, DNLP, NLP, RMINLP, QCP, RMIQCP, MIQCP, CNS }

The above statement should appear before the Solve statement. If Couenne was specified as the default solver during GAMS installation, the above statement is not necessary.

1.1 Specification of Options

A Couenne option file contains IPOPT, BONMIN, and Couenne options, for clarity all BONMIN options should be preceded with the prefix bonmin and all Couenne options should be preceded with the prefix couenne. All IPOPT and many BONMIN options are available in Couenne.

The scheme to name option files is the same as for all other GAMS solvers. The format of the option file is the same as for IPOPT.

GAMS/Couenne understands currently the following GAMS parameters: reslim (time limit), nodlim (node limit), cutoff, optca (absolute gap tolerance), and optcr (relative gap tolerance).

Further, the option threads can be used to control the number of threads used in the linear algebra routines of IPOPT, see The linear solver in IPOPT in the IPOPT manual for details.

2 List of Options

In the following we give a list of options available for Couenne, including those for the underlying Ipopt and Bonmin solvers.

2.1 Couenne options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>2mir_cuts</td>
<td>Frequency k (in terms of nodes) for generating 2mir_cuts cuts in branch-and-cut.</td>
<td>0</td>
</tr>
<tr>
<td>aggressive_fbbt</td>
<td>Aggressive feasibility-based bound tightening (to use with NLP points)</td>
<td>yes</td>
</tr>
<tr>
<td>art_cutoff</td>
<td>Artificial cutoff</td>
<td>maxdouble</td>
</tr>
<tr>
<td>art_lower</td>
<td>Artificial lower bound</td>
<td>mindouble</td>
</tr>
<tr>
<td>boundtightening_print_level</td>
<td>Output level for bound tightening code in Couenne</td>
<td>0</td>
</tr>
<tr>
<td>branching_object</td>
<td>type of branching object for variable selection</td>
<td>var_obj</td>
</tr>
<tr>
<td>branching_print_level</td>
<td>Output level for branching code in Couenne</td>
<td>0</td>
</tr>
<tr>
<td>branch_conv_cuts</td>
<td>Apply convexification cuts before branching (for now only within strong branching)</td>
<td>yes</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>branch_fbbt</td>
<td>Apply bound tightening before branching</td>
<td>yes</td>
</tr>
<tr>
<td>branch_lpb</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_cube</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_div</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_exp</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_log</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_negpow</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_pow</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_prod</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_sqr</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_lpb_trig</td>
<td>Defines safe interval percentage [0,0.5] for using LP point as a branching point.</td>
<td>0.2</td>
</tr>
<tr>
<td>branch_midpoint_alpha</td>
<td>Defines convex combination of mid point and current LP point: b = alpha x_{lp} + (1-alpha) (lb+ub)/2.</td>
<td>0.25</td>
</tr>
<tr>
<td>branch_pt_select</td>
<td>Chooses branching point selection strategy</td>
<td>mid-point</td>
</tr>
<tr>
<td>branch_pt_select_cube</td>
<td>Chooses branching point selection strategy for operator cube.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_div</td>
<td>Chooses branching point selection strategy for operator div.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_exp</td>
<td>Chooses branching point selection strategy for operator exp.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_log</td>
<td>Chooses branching point selection strategy for operator log.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_negpow</td>
<td>Chooses branching point selection strategy for operator negpow.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_pow</td>
<td>Chooses branching point selection strategy for operator pow.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_prod</td>
<td>Chooses branching point selection strategy for operator prod.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_sqr</td>
<td>Chooses branching point selection strategy for operator sqr.</td>
<td>common</td>
</tr>
<tr>
<td>branch_pt_select_trig</td>
<td>Chooses branching point selection strategy for operator trig.</td>
<td>common</td>
</tr>
<tr>
<td>check_lpb</td>
<td>Check all LPs through an independent call to OsiClpSolverInterface::initialSolve()</td>
<td>no</td>
</tr>
<tr>
<td>clique_cuts</td>
<td>Frequency k (in terms of nodes) for generating clique_cuts cuts in branch-and-cut.</td>
<td>0</td>
</tr>
<tr>
<td>cont_var_priority</td>
<td>Priority of continuous variable branching</td>
<td>2000</td>
</tr>
<tr>
<td>convexification_cuts</td>
<td>Specify the frequency (in terms of nodes) at which couenne ecp cuts are generated.</td>
<td>1</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>convexification_points</code></td>
<td>Specify the number of points at which to convexify when convexification type is uniform-grid or around-current-point.</td>
<td>4</td>
</tr>
<tr>
<td><code>convexification_type</code></td>
<td>Determines in which point the linear over/under-estimator are generated</td>
<td>current-point-only</td>
</tr>
<tr>
<td><code>convexifying_print_level</code></td>
<td>Output level for convexifying code in Couenne</td>
<td>0</td>
</tr>
<tr>
<td><code>cover_cuts</code></td>
<td>Frequency k (in terms of nodes) for generating cover_cuts cuts in branch-and-cut.</td>
<td>0</td>
</tr>
<tr>
<td><code>delete_redundant</code></td>
<td>Eliminate redundant variables, which appear in the problem as x_k = x_h</td>
<td>yes</td>
</tr>
<tr>
<td><code>disjcuts_print_level</code></td>
<td>Output level for disjunctive cuts in Couenne</td>
<td>0</td>
</tr>
<tr>
<td><code>disj_active_cols</code></td>
<td>Only include violated variable bounds in the Cut Generating LP (CGLP).</td>
<td>no</td>
</tr>
<tr>
<td><code>disj_active_rows</code></td>
<td>Only include violated linear inequalities in the CGLP.</td>
<td>no</td>
</tr>
<tr>
<td><code>disj_cumulative</code></td>
<td>Add previous disjunctive cut to current CGLP.</td>
<td>no</td>
</tr>
<tr>
<td><code>disj_depth_level</code></td>
<td>Depth of the B&amp;B tree when to start decreasing the number of objects that generate disjunctions.</td>
<td>5</td>
</tr>
<tr>
<td><code>disj_depth_stop</code></td>
<td>Depth of the B&amp;B tree where separation of disjunctive cuts is stopped.</td>
<td>20</td>
</tr>
<tr>
<td><code>disj_init_number</code></td>
<td>Maximum number of disjunction to consider at each iteration.</td>
<td>10</td>
</tr>
<tr>
<td><code>disj_init_perc</code></td>
<td>The maximum fraction of all disjunctions currently violated by the problem to consider for generating disjunctions.</td>
<td>0.5</td>
</tr>
<tr>
<td><code>display_stats</code></td>
<td>display statistics at the end of the run</td>
<td>no</td>
</tr>
<tr>
<td><code>enable_lp_implied_bounds</code></td>
<td>Enable OsiSolverInterface::tightenBounds () – warning: it has caused some trouble to Couenne</td>
<td>no</td>
</tr>
<tr>
<td><code>enable_sos</code></td>
<td>Use Special Ordered Sets (SOS) as indicated in the MINLP model</td>
<td>no</td>
</tr>
<tr>
<td><code>estimate_select</code></td>
<td>How the min/max estimates of the subproblems’ bounds are used in strong branching</td>
<td>normal</td>
</tr>
<tr>
<td><code>feasibility_bt</code></td>
<td>Feasibility-based (cheap) bound tightening (FBBT)</td>
<td>yes</td>
</tr>
<tr>
<td><code>feas_pump_convcuts</code></td>
<td>Separate MILP-feasible, MINLP-infeasible solution during or after MILP solver.</td>
<td>none</td>
</tr>
<tr>
<td><code>feas_pump_heuristic</code></td>
<td>Apply the nonconvex Feasibility Pump</td>
<td>no</td>
</tr>
<tr>
<td><code>feas_pump_iter</code></td>
<td>Number of iterations in the main Feasibility Pump loop</td>
<td>10</td>
</tr>
<tr>
<td><code>feas_pump_level</code></td>
<td>Specify the logarithm of the number of feasibility pumps to perform on average for each level of given depth of the tree.</td>
<td>3</td>
</tr>
<tr>
<td><code>feas_pump_milpmethod</code></td>
<td>How should the integral solution be constructed?</td>
<td>-1</td>
</tr>
<tr>
<td><code>feas_pump_mult_dist_milp</code></td>
<td>Weight of the distance in the distance function of the milp problem</td>
<td>0</td>
</tr>
<tr>
<td><code>feas_pump_mult_dist_nlp</code></td>
<td>Weight of the distance in the distance function of the nlp problem</td>
<td>0</td>
</tr>
<tr>
<td><code>feas_pump_mult_hess_milp</code></td>
<td>Weight of the Hessian in the distance function of the milp problem</td>
<td>0</td>
</tr>
<tr>
<td><code>feas_pump_mult_hess_nlp</code></td>
<td>Weight of the Hessian in the distance function of the nlp problem</td>
<td>0</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>feas_pump_mult_objf_milp</td>
<td>Weight of the original objective function in the distance function of the milp problem</td>
<td>0</td>
</tr>
<tr>
<td>feas_pump_mult_objf_nlp</td>
<td>Weight of the original objective function in the distance function of the nlp problem</td>
<td>0</td>
</tr>
<tr>
<td>feas_pump_nseprounds</td>
<td>Number of rounds that separate convexification cuts. Must be at least 1</td>
<td>4</td>
</tr>
<tr>
<td>feas_pump_poolcomp</td>
<td>Priority field to compare solutions in FP pool</td>
<td>0</td>
</tr>
<tr>
<td>feas_pump_tabumgt</td>
<td>Retrieval of MILP solutions when the one returned is unsatisfactory</td>
<td>pool</td>
</tr>
<tr>
<td>feas_pump_usescip</td>
<td>Should SCIP be used to solve the MILPs?</td>
<td>yes</td>
</tr>
<tr>
<td>feas_pump_vardist</td>
<td>Distance computed on integer-only or on both types of variables, in different flavors</td>
<td>integer</td>
</tr>
<tr>
<td>feas_tolerance</td>
<td>Tolerance for constraints/auxiliary variables</td>
<td>1e-05</td>
</tr>
<tr>
<td>fixpoint_bt</td>
<td>The frequency (in terms of nodes) at which Fix Point Bound Tightening is performed</td>
<td>0</td>
</tr>
<tr>
<td>fixpoint_bt_model</td>
<td>Choose whether to add an extended fixpoint LP model or a more compact one.</td>
<td>compact</td>
</tr>
<tr>
<td>flow_covers_cuts</td>
<td>Frequency k (in terms of nodes) for generating flow_covers_cuts cuts in branch-and-cut</td>
<td>0</td>
</tr>
<tr>
<td>Gomory_cuts</td>
<td>Frequency k (in terms of nodes) for generating Gomory_cuts cuts in branch-and-cut</td>
<td>0</td>
</tr>
<tr>
<td>int_var_priority</td>
<td>Priority of integer variable branching</td>
<td>1000</td>
</tr>
<tr>
<td>iterative_rounding_aggressiveness</td>
<td>Aggressiveness of the Iterative Rounding heuristic</td>
<td>1</td>
</tr>
<tr>
<td>iterative_rounding_base_lbrhs</td>
<td>Base rhs of the local branching constraint for Iterative Rounding</td>
<td>15</td>
</tr>
<tr>
<td>iterative_rounding_heuristic</td>
<td>Do we use the Iterative Rounding heuristic</td>
<td>no</td>
</tr>
<tr>
<td>iterative_rounding_num_fir_points</td>
<td>Max number of points rounded at the beginning of Iterative Rounding</td>
<td>5</td>
</tr>
<tr>
<td>iterative_rounding_omega</td>
<td>Omega parameter of the Iterative Rounding heuristic</td>
<td>0.2</td>
</tr>
<tr>
<td>iterative_rounding_time</td>
<td>Specify the maximum time allowed for the Iterative Rounding heuristic</td>
<td>-1</td>
</tr>
<tr>
<td>iterative_rounding_time_firstcall</td>
<td>Specify the maximum time allowed for the Iterative Rounding heuristic when no feasible solution is known</td>
<td>-1</td>
</tr>
<tr>
<td>lift_and_project_cuts</td>
<td>Frequency k (in terms of nodes) for generating lift_and_project_cuts cuts in branch-and-cut</td>
<td>0</td>
</tr>
<tr>
<td>local_branching_heuristic</td>
<td>Apply local branching heuristic</td>
<td>no</td>
</tr>
<tr>
<td>local_optimization_heuristic</td>
<td>Search for local solutions of MINLPs</td>
<td>yes</td>
</tr>
<tr>
<td>log_num_abt_per_level</td>
<td>Specify the frequency (in terms of nodes) for aggressive bound tightening.</td>
<td>2</td>
</tr>
<tr>
<td>log_num_local_optimization_per_level</td>
<td>Specify the logarithm of the number of local optimizations to perform on average for each level of given depth of the tree</td>
<td>2</td>
</tr>
<tr>
<td>log_num_obbt_per_level</td>
<td>Specify the frequency (in terms of nodes) for optimality-based bound tightening</td>
<td>1</td>
</tr>
<tr>
<td>lp_solver</td>
<td>Linear Programming solver for the linearization</td>
<td>clp</td>
</tr>
<tr>
<td>max_fbbt_iter</td>
<td>Number of FBBT iterations before stopping even with tightened bounds.</td>
<td>3</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>minlp_disj_cuts</td>
<td>The frequency (in terms of nodes) at which Couenne disjunctive cuts are generated.</td>
<td>0</td>
</tr>
<tr>
<td>mir_cuts</td>
<td>Frequency k (in terms of nodes) for generating mir_cuts cuts in branch-and-cut.</td>
<td>0</td>
</tr>
<tr>
<td>multilinear_separation</td>
<td>Separation for multilinear terms</td>
<td>tight</td>
</tr>
<tr>
<td>nlpheur_print_level</td>
<td>Output level for NLP heuristic in Couenne</td>
<td>0</td>
</tr>
<tr>
<td>optimality_bt</td>
<td>Optimality-based (expensive) bound tightening (OBBT)</td>
<td>yes</td>
</tr>
<tr>
<td>orbital_branching</td>
<td>detect symmetries and apply orbital branching</td>
<td>no</td>
</tr>
<tr>
<td>output_level</td>
<td>Output level</td>
<td>0</td>
</tr>
<tr>
<td>probing_cuts</td>
<td>Frequency k (in terms of nodes) for generating probing_cuts cuts in branch-and-cut.</td>
<td>0</td>
</tr>
<tr>
<td>problem_print_level</td>
<td>Output level for problem manipulation code in Couenne</td>
<td>2</td>
</tr>
<tr>
<td>pseudocost_mult</td>
<td>Multipliers of pseudocosts for estimating and update estimation of bound</td>
<td>interval_br_rev</td>
</tr>
<tr>
<td>pseudocost_mult_lp</td>
<td>Use distance between LP points to update multipliers of pseudocosts after simulating branching</td>
<td>no</td>
</tr>
<tr>
<td>quadrilinear_decomp</td>
<td>type of decomposition for quadrilinear terms (see work by Cafieri, Lee, Liberti)</td>
<td>rAI</td>
</tr>
<tr>
<td>redcost_bt</td>
<td>Reduced cost bound tightening</td>
<td>yes</td>
</tr>
<tr>
<td>reduce_split_cuts</td>
<td>Frequency k (in terms of nodes) for generating reduce_split_cuts cuts in branch-and-cut.</td>
<td>0</td>
</tr>
<tr>
<td>red_cost_branching</td>
<td>Apply Reduced Cost Branching (instead of the Violation Transfer) – MUST have vt_obj enabled</td>
<td>no</td>
</tr>
<tr>
<td>reformulate_print_level</td>
<td>Output level for reformulating problems in Couenne</td>
<td>0</td>
</tr>
<tr>
<td>solvetrace</td>
<td>Name of file for writing solving progress information.</td>
<td></td>
</tr>
<tr>
<td>solvetracenodefreq</td>
<td>Frequency in number of nodes for writing solving progress information.</td>
<td>100</td>
</tr>
<tr>
<td>solvetrace_timefreq</td>
<td>Frequency in seconds for writing solving progress information.</td>
<td>5</td>
</tr>
<tr>
<td>trust_strong</td>
<td>Fathom strong branching LPs when their bound is above the cutoff</td>
<td>yes</td>
</tr>
<tr>
<td>twoimpl_depth_level</td>
<td>Depth of the B&amp;B tree when to start decreasing the chance of running this algorithm.</td>
<td>5</td>
</tr>
<tr>
<td>twoimpl_depth_stop</td>
<td>Depth of the B&amp;B tree where separation is stopped.</td>
<td>20</td>
</tr>
<tr>
<td>two_implied_bt</td>
<td>The frequency (in terms of nodes) at which Couenne two-implied bounds are tightened.</td>
<td>0</td>
</tr>
<tr>
<td>two_implied_max_trials</td>
<td>The number of iteration at each call to the cut generator.</td>
<td>2</td>
</tr>
<tr>
<td>use_auxcons</td>
<td>Use constraints-defined auxiliaries, i.e. auxiliaries w = f(x) defined by original constraints f(x) - w = 0</td>
<td>yes</td>
</tr>
<tr>
<td>use_quadratic</td>
<td>Use quadratic expressions and related exprQuad class</td>
<td>no</td>
</tr>
<tr>
<td>use_semiiaux</td>
<td>Use semiauxiliaries, i.e. auxiliaries defined as w ≥ f(x) rather than w := f(x))</td>
<td>yes</td>
</tr>
<tr>
<td>violated_cuts_only</td>
<td>Yes if only violated convexification cuts should be added</td>
<td>yes</td>
</tr>
</tbody>
</table>
2.2 Bonmin Branch-and-bound options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowable_fraction_gap</td>
<td>Specify the value of relative gap under which the algorithm stops.</td>
<td>0.1</td>
</tr>
<tr>
<td>allowable_gap</td>
<td>Specify the value of absolute gap under which the algorithm stops.</td>
<td>0</td>
</tr>
<tr>
<td>cutoff</td>
<td>Specify cutoff value.</td>
<td>1e+100</td>
</tr>
<tr>
<td>cutoff_decr</td>
<td>Specify cutoff decrement.</td>
<td>1e-05</td>
</tr>
<tr>
<td>enable_dynamic_nlp</td>
<td>Enable dynamic linear and quadratic rows addition in nlp</td>
<td>no</td>
</tr>
<tr>
<td>integer_tolerance</td>
<td>Set integer tolerance.</td>
<td>1e-06</td>
</tr>
<tr>
<td>iteration_limit</td>
<td>Set the cumulative maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound.</td>
<td>maxint</td>
</tr>
<tr>
<td>nlp_failure_behavior</td>
<td>Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).</td>
<td>stop</td>
</tr>
<tr>
<td>node_comparison</td>
<td>Choose the node selection strategy.</td>
<td>best-bound</td>
</tr>
<tr>
<td>node_limit</td>
<td>Set the maximum number of nodes explored in the branch-and-bound search.</td>
<td>maxint</td>
</tr>
<tr>
<td>number_before_trust</td>
<td>Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching.</td>
<td>8</td>
</tr>
<tr>
<td>number_strong_branch</td>
<td>Choose the maximum number of variables considered for strong branching.</td>
<td>20</td>
</tr>
<tr>
<td>num_cut_passes</td>
<td>Set the maximum number of cut passes at regular nodes of the branch-and-cut.</td>
<td>1</td>
</tr>
<tr>
<td>num_cut_passes_at_root</td>
<td>Set the maximum number of cut passes at regular nodes of the branch-and-cut.</td>
<td>20</td>
</tr>
<tr>
<td>random_generator_seed</td>
<td>Set seed for random number generator (a value of -1 sets seeds to time since Epoch).</td>
<td>0</td>
</tr>
<tr>
<td>read_solution_file</td>
<td>Read a file with the optimal solution to test if algorithms cuts it.</td>
<td>no</td>
</tr>
<tr>
<td>solution_limit</td>
<td>Abort after that much integer feasible solution have been found by algorithm</td>
<td>maxint</td>
</tr>
<tr>
<td>time_limit</td>
<td>Set the global maximum computation time (in secs) for the algorithm.</td>
<td>1000</td>
</tr>
<tr>
<td>tree_search_strategy</td>
<td>Pick a strategy for traversing the tree</td>
<td>probed-dive</td>
</tr>
<tr>
<td>variable_selection</td>
<td>Chooses variable selection strategy</td>
<td>strong-branching</td>
</tr>
</tbody>
</table>

2.3 Bonmin NLP interface

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>warm_start</td>
<td>Select the warm start method</td>
<td>none</td>
</tr>
</tbody>
</table>

2.4 Bonmin NLP solution robustness

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>Choice of the algorithm.</td>
<td>B-BB</td>
</tr>
<tr>
<td>max_consecutive_failures</td>
<td>(temporarily removed) Number n of consecutive unsolved problems before aborting a branch of the tree.</td>
<td>10</td>
</tr>
<tr>
<td>max_random_point_radius</td>
<td>Set max value r for coordinate of a random point.</td>
<td>100000</td>
</tr>
<tr>
<td>milp_solver</td>
<td>Choose the subsolver to solve MILP sub-problems in OA decompositions.</td>
<td>Cbc_D</td>
</tr>
</tbody>
</table>
### milp_strategy
Choose a strategy for MILPs.

#### num_iterations_suspect
Number of iterations over which a node is considered 'suspect' (for debugging purposes only, see detailed documentation).

-1

#### num_retry_unsolved_random_point
Number of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).
0

#### random_point_perturbation_interval
Amount by which starting point is perturbed when choosing to pick random point by perturbing starting point
1

#### random_point_type
Method to choose a random starting point
Jon

#### resolve_on_small_infeasibility
If a locally infeasible problem is infeasible by less than this, resolve it with initial starting point.
0

### 2.5 Bonmin Nonconvex problems

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>coeff_var_threshold</td>
<td>Coefficient of variation threshold (for dynamic definition of cutoff_decr).</td>
<td>0.1</td>
</tr>
<tr>
<td>dynamic_def_cutoff_decr</td>
<td>Do you want to define the parameter cutoff_decr dynamically?</td>
<td>no</td>
</tr>
<tr>
<td>first_perc_for_cutoff_decr</td>
<td>The percentage used when, the coeff of variance is smaller than the threshold, to compute the cutoff_decr dynamically.</td>
<td>-0.02</td>
</tr>
<tr>
<td>max_consecutive_infeasible</td>
<td>Number of consecutive infeasible subproblems before aborting a branch.</td>
<td>0</td>
</tr>
<tr>
<td>num_resolve_at_infeasibles</td>
<td>Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point.</td>
<td>0</td>
</tr>
<tr>
<td>num_resolve_at_node</td>
<td>Number k of tries to resolve a node (other than the root) of the tree with different starting point.</td>
<td>0</td>
</tr>
<tr>
<td>num_resolve_at_root</td>
<td>Number k of tries to resolve the root node with different starting points.</td>
<td>0</td>
</tr>
<tr>
<td>second_perc_for_cutoff_decr</td>
<td>The percentage used when, the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically.</td>
<td>-0.05</td>
</tr>
</tbody>
</table>

### 2.6 Bonmin Outer Approximation cuts generation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_only_violated_oa</td>
<td>Do we add all OA cuts or only the ones violated by current point?</td>
<td>no</td>
</tr>
<tr>
<td>oa_cuts_scope</td>
<td>Specify if OA cuts added are to be set globally or locally valid</td>
<td>global</td>
</tr>
<tr>
<td>oa_rhs_relax</td>
<td>Value by which to relax OA cut</td>
<td>1e-08</td>
</tr>
<tr>
<td>tiny_element</td>
<td>Value for tiny element in OA cut</td>
<td>1e-08</td>
</tr>
<tr>
<td>very_tiny_element</td>
<td>Value for very tiny element in OA cut</td>
<td>1e-17</td>
</tr>
</tbody>
</table>

### 2.7 Bonmin Output and Loglevel

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb_log_interval</td>
<td>Interval at which node level output is printed.</td>
<td>100</td>
</tr>
<tr>
<td>bb_log_level</td>
<td>Specify main branch-and-bound log level.</td>
<td>1</td>
</tr>
<tr>
<td>lp_log_level</td>
<td>Specify LP log level.</td>
<td>0</td>
</tr>
<tr>
<td>nlp_log_at_root</td>
<td>Specify a different log level for root relaxation.</td>
<td>0</td>
</tr>
</tbody>
</table>
2.8 Bonmin Strong branching setup

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidate_sort_criterion</td>
<td>Choice of the criterion to choose candidates in strong-branching</td>
<td>best-ps-cost</td>
</tr>
<tr>
<td>maxmin_crit_have_sol</td>
<td>Weight towards minimum in of lower and upper branching estimates when a solution has been found.</td>
<td>0.1</td>
</tr>
<tr>
<td>maxmin_crit_no_sol</td>
<td>Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.</td>
<td>0.7</td>
</tr>
<tr>
<td>min_number_strong_branch</td>
<td>Sets minimum number of variables for strong branching (overriding trust)</td>
<td>0</td>
</tr>
<tr>
<td>number_before_trust_list</td>
<td>Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list.</td>
<td>0</td>
</tr>
<tr>
<td>number_look_ahead</td>
<td>Sets limit of look-ahead strong-branching trials</td>
<td>0</td>
</tr>
<tr>
<td>number_strong_branch_root</td>
<td>Maximum number of variables considered for strong branching in root node.</td>
<td>maxint</td>
</tr>
<tr>
<td>setup_pseudo_frac</td>
<td>Proportion of strong branching list that has to be taken from most-integer-infeasible list.</td>
<td>0.5</td>
</tr>
<tr>
<td>trust_strong_branching_for_pseudo_cost</td>
<td>Whether or not to trust strong branching results for updating pseudo costs.</td>
<td>yes</td>
</tr>
</tbody>
</table>

2.9 Ipopt Barrier Parameter Update

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptive_mu_globalization</td>
<td>Globalization strategy for the adaptive mu selection mode.</td>
<td>obj-constr-filter</td>
</tr>
<tr>
<td>adaptive_mu_kkterror_red_fact</td>
<td>Sufficient decrease factor for <code>kkt-error</code> globalization strategy.</td>
<td>0.9999</td>
</tr>
<tr>
<td>adaptive_mu_kkterror_red_itors</td>
<td>Maximum number of iterations requiring sufficient progress.</td>
<td>4</td>
</tr>
<tr>
<td>adaptive_mu_kkt_norm_type</td>
<td>Norm used for the KKT error in the adaptive mu globalization strategies.</td>
<td>2-norm-squared</td>
</tr>
<tr>
<td>adaptive_mu_monotone_init_factor</td>
<td>Determines the initial value of the barrier parameter when switching to the monotone mode.</td>
<td>0.8</td>
</tr>
<tr>
<td>adaptive_mu_restore_previous_iterate</td>
<td>Indicates if the previous iterate should be restored if the monotone mode is entered.</td>
<td>no</td>
</tr>
<tr>
<td>barrier_tol_factor</td>
<td>Factor for mu in barrier stop test.</td>
<td>10</td>
</tr>
<tr>
<td>filter_margin_fact</td>
<td>Factor determining width of margin for obj-constr-filter adaptive globalization strategy.</td>
<td>1e-05</td>
</tr>
<tr>
<td>filter_max_margin</td>
<td>Maximum width of margin in obj-constr-filter adaptive globalization strategy.</td>
<td>1</td>
</tr>
<tr>
<td>fixed_mu_oracle</td>
<td>Oracle for the barrier parameter when switching to fixed mode.</td>
<td>average_compl</td>
</tr>
</tbody>
</table>
### mu_allow_fast_monotone_decrease
Allow skipping of barrier problem if barrier test is already met.  yes

### mu_init
Initial value for the barrier parameter.  0.1

### mu_linear_decrease_factor
Determines linear decrease rate of barrier parameter.  0.2

### mu_max
Maximum value for barrier parameter.  100000

### mu_max_fact
Factor for initialization of maximum value for barrier parameter.  1000

### mu_min
Minimum value for barrier parameter.  1e-11

### mu_oracle
Oracle for a new barrier parameter in the adaptive strategy.  quality-function

### mu_strategy
Update strategy for barrier parameter.  monotone

### mu_superlinear_decrease_power
Determines superlinear decrease rate of barrier parameter.  1.5

### quality_function_balancing_term
The balancing term included in the quality function for centrality.  none

### quality_function_centrality
The penalty term for centrality that is included in quality function.  none

### quality_function_max_section_steps
Maximum number of search steps during direct search procedure determining the optimal centering parameter.  8

### quality_function_norm_type
Norm used for components of the quality function.  2-norm-squared

### quality_function_section_qf_tol
Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).  0

### quality_function_section_sigma_tol
Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).  0.01

### sigma_max
Maximum value of the centering parameter.  100

### sigma_min
Minimum value of the centering parameter.  1e-06

### tau_min
Lower bound on fraction-to-the-boundary parameter tau.  0.99

### 2.10 Ipopt Convergence

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptable_compl_inf_tol</td>
<td>’Acceptance’ threshold for the complementarity conditions.</td>
<td>0.01</td>
</tr>
<tr>
<td>acceptable_constr_viol_tol</td>
<td>’Acceptance’ threshold for the constraint violation.</td>
<td>0.01</td>
</tr>
<tr>
<td>acceptable_dual_inf_tol</td>
<td>’Acceptance’ threshold for the dual infeasibility.</td>
<td>1e+10</td>
</tr>
<tr>
<td>acceptable_iter</td>
<td>Number of ‘acceptable’ iterates before triggering termination.</td>
<td>15</td>
</tr>
<tr>
<td>acceptable_obj_change_tol</td>
<td>’Acceptance’ stopping criterion based on objective function change.</td>
<td>1e+20</td>
</tr>
<tr>
<td>acceptable_tol</td>
<td>’Acceptance’ convergence tolerance (relative).</td>
<td>1e-06</td>
</tr>
<tr>
<td>compl_inf_tol</td>
<td>Desired threshold for the complementarity conditions.</td>
<td>0.0001</td>
</tr>
<tr>
<td>constr_viol_tol</td>
<td>Desired threshold for the constraint violation.</td>
<td>0.0001</td>
</tr>
<tr>
<td>diverging_iterates_tol</td>
<td>Threshold for maximal value of primal iterates.</td>
<td>1e+20</td>
</tr>
<tr>
<td>dual_inf_tol</td>
<td>Desired threshold for the dual infeasibility.</td>
<td>1</td>
</tr>
<tr>
<td>max_cpu_time</td>
<td>Maximum number of CPU seconds.</td>
<td>1e+06</td>
</tr>
<tr>
<td>max_iter</td>
<td>Maximum number of iterations.</td>
<td>3000</td>
</tr>
<tr>
<td>mu_target</td>
<td>Desired value of complementarity.</td>
<td>0</td>
</tr>
<tr>
<td>s_max</td>
<td>Scaling threshold for the NLP error.</td>
<td>100</td>
</tr>
</tbody>
</table>
tol | Desired convergence tolerance (relative). | 1e-08

### 2.11 Ipopt Hessian Approximation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian_approximation</td>
<td>Indicates what Hessian information is to be used.</td>
<td>exact</td>
</tr>
<tr>
<td>hessian_approximation_space</td>
<td>Indicates in which subspace the Hessian information is to be approximated.</td>
<td>nonlinear-variables</td>
</tr>
<tr>
<td>limited_memory_aug_solver</td>
<td>Strategy for solving the augmented system for low-rank Hessian.</td>
<td>sherman-morrison</td>
</tr>
<tr>
<td>limited_memory_initialization</td>
<td>Initialization strategy for the limited memory quasi-Newton approximation.</td>
<td>scalar1</td>
</tr>
<tr>
<td>limited_memory_init_val</td>
<td>Value for B0 in low-rank update.</td>
<td>1</td>
</tr>
<tr>
<td>limited_memory_init_val_max</td>
<td>Upper bound on value for B0 in low-rank update.</td>
<td>1e+08</td>
</tr>
<tr>
<td>limited_memory_init_val_min</td>
<td>Lower bound on value for B0 in low-rank update.</td>
<td>1e-08</td>
</tr>
<tr>
<td>limited_memory_max_history</td>
<td>Maximum size of the history for the limited quasi-Newton Hessian approximation.</td>
<td>6</td>
</tr>
<tr>
<td>limited_memory_max_skipping</td>
<td>Threshold for successive iterations where update is skipped.</td>
<td>2</td>
</tr>
<tr>
<td>limited_memory_special_for_resto</td>
<td>Determines if the quasi-Newton updates should be special during the restoration phase.</td>
<td>no</td>
</tr>
<tr>
<td>limited_memory_update_type</td>
<td>Quasi-Newton update formula for the limited memory approximation.</td>
<td>bfgs</td>
</tr>
</tbody>
</table>

### 2.12 Ipopt Initialization

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound_frac</td>
<td>Desired minimum relative distance from the initial point to bound.</td>
<td>0.01</td>
</tr>
<tr>
<td>bound_mult_init_method</td>
<td>Initialization method for bound multipliers</td>
<td>constant</td>
</tr>
<tr>
<td>bound_mult_init_val</td>
<td>Initial value for the bound multipliers.</td>
<td>1</td>
</tr>
<tr>
<td>bound_push</td>
<td>Desired minimum absolute distance from the initial point to bound.</td>
<td>0.01</td>
</tr>
<tr>
<td>constr_mult_init_max</td>
<td>Maximum allowed least-square guess of constraint multipliers.</td>
<td>1000</td>
</tr>
<tr>
<td>least_square_init_duals</td>
<td>Least square initialization of all dual variables</td>
<td>no</td>
</tr>
<tr>
<td>least_square_init primal</td>
<td>Least square initialization of the primal variables</td>
<td>no</td>
</tr>
<tr>
<td>slack_bound_frac</td>
<td>Desired minimum relative distance from the initial slack to bound.</td>
<td>0.01</td>
</tr>
<tr>
<td>slack_bound_push</td>
<td>Desired minimum absolute distance from the initial slack to bound.</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### 2.13 Ipopt Line Search

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_after_max_steps</td>
<td>Accept a trial point after maximal this number of steps.</td>
<td>-1</td>
</tr>
<tr>
<td>accept_every_trial_step</td>
<td>Always accept the first trial step.</td>
<td>no</td>
</tr>
<tr>
<td>alpha_for_y</td>
<td>Method to determine the step size for constraint multipliers.</td>
<td>primal</td>
</tr>
<tr>
<td>alpha_for_y_tol</td>
<td>Tolerance for switching to full equality multiplier steps.</td>
<td>10</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>alpha_min_frac</td>
<td>Safety factor for the minimal step size (before switching to restoration phase).</td>
<td>0.05</td>
</tr>
<tr>
<td>alpha_red_factor</td>
<td>Fractional reduction of the trial step size in the backtracking line search.</td>
<td>0.5</td>
</tr>
<tr>
<td>constraintViolation_norm_type</td>
<td>Norm to be used for the constraint violation in the line search.</td>
<td>1-norm</td>
</tr>
<tr>
<td>corrector_compl_avg_red_fact</td>
<td>Complementarity tolerance factor for accepting corrector step (unsupported!).</td>
<td>1</td>
</tr>
<tr>
<td>corrector_type</td>
<td>The type of corrector steps that should be taken (unsupported!).</td>
<td>none</td>
</tr>
<tr>
<td>delta</td>
<td>Multiplier for constraint violation in the switching rule.</td>
<td>1</td>
</tr>
<tr>
<td>eta_phi</td>
<td>Relaxation factor in the Armijo condition.</td>
<td>1e-08</td>
</tr>
<tr>
<td>filter_reset_trigger</td>
<td>Number of iterations that trigger the filter reset.</td>
<td>5</td>
</tr>
<tr>
<td>gamma_phi</td>
<td>Relaxation factor in the filter margin for the barrier function.</td>
<td>1e-08</td>
</tr>
<tr>
<td>gamma_theta</td>
<td>Relaxation factor in the filter margin for the constraint violation.</td>
<td>1e-05</td>
</tr>
<tr>
<td>kappa_sigma</td>
<td>Factor limiting the deviation of dual variables from primal estimates.</td>
<td>1e+10</td>
</tr>
<tr>
<td>kappa_soc</td>
<td>Factor in the sufficient reduction rule for second order correction.</td>
<td>0.99</td>
</tr>
<tr>
<td>line_search_method</td>
<td>Globalization method used in backtracking line search</td>
<td>filter</td>
</tr>
<tr>
<td>max_filter_resets</td>
<td>Maximal allowed number of filter resets</td>
<td>5</td>
</tr>
<tr>
<td>max_soc</td>
<td>Maximum number of second order correction trial steps at each iteration.</td>
<td>4</td>
</tr>
<tr>
<td>nu_inc</td>
<td>Increment of the penalty parameter.</td>
<td>0.0001</td>
</tr>
<tr>
<td>nu_init</td>
<td>Initial value of the penalty parameter.</td>
<td>1e-06</td>
</tr>
<tr>
<td>obj_max_inc</td>
<td>Determines the upper bound on the acceptable increase of barrier objective function.</td>
<td>5</td>
</tr>
<tr>
<td>recalc_y</td>
<td>Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.</td>
<td>no</td>
</tr>
<tr>
<td>recalc_y_feas_tol</td>
<td>Feasibility threshold for recomputation of multipliers.</td>
<td>1e-06</td>
</tr>
<tr>
<td>rho</td>
<td>Value in penalty parameter update formula.</td>
<td>0.1</td>
</tr>
<tr>
<td>skip_corr_if_neg_curv</td>
<td>Skip the corrector step in negative curvature iteration (unsupported!).</td>
<td>yes</td>
</tr>
<tr>
<td>skip_corr_in_monotone_mode</td>
<td>Skip the corrector step during monotone barrier parameter mode (unsupported!).</td>
<td>yes</td>
</tr>
<tr>
<td>slack_move</td>
<td>Correction size for very small slacks.</td>
<td>1.81899e-12</td>
</tr>
<tr>
<td>s_phi</td>
<td>Exponent for linear barrier function model in the switching rule.</td>
<td>2.3</td>
</tr>
<tr>
<td>s_theta</td>
<td>Exponent for current constraint violation in the switching rule.</td>
<td>1.1</td>
</tr>
<tr>
<td>theta_max_fact</td>
<td>Determines upper bound for constraint violation in the filter.</td>
<td>10000</td>
</tr>
<tr>
<td>theta_min_fact</td>
<td>Determines constraint violation threshold in the switching rule.</td>
<td>0.0001</td>
</tr>
<tr>
<td>tiny_step_tol</td>
<td>Tolerance for detecting numerically insignificant steps.</td>
<td>2.22045e-15</td>
</tr>
<tr>
<td>tiny_step_y_tol</td>
<td>Tolerance for quitting because of numerically insignificant steps.</td>
<td>0.01</td>
</tr>
<tr>
<td>watchdog_shortened_iter_trigger</td>
<td>Number of shortened iterations that trigger the watchdog.</td>
<td>10</td>
</tr>
<tr>
<td>watchdog_trial_iter_max</td>
<td>Maximum number of watchdog iterations.</td>
<td>3</td>
</tr>
</tbody>
</table>

## 2.14 Ipopt Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear_scaling_on_demand</td>
<td>Flag indicating that linear scaling is only done if it seems required.</td>
<td>yes</td>
</tr>
<tr>
<td>linear_solver</td>
<td>Linear solver used for step computations.</td>
<td>ma27</td>
</tr>
</tbody>
</table>
### 2.15 Ipopt MA27 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma27_ignore_singularity</td>
<td>Enables MA27’s ability to solve a linear system even if the matrix is singular.</td>
<td>no</td>
</tr>
<tr>
<td>ma27_la_init_factor</td>
<td>Real workspace memory for MA27.</td>
<td>5</td>
</tr>
<tr>
<td>ma27_liw_init_factor</td>
<td>Integer workspace memory for MA27.</td>
<td>5</td>
</tr>
<tr>
<td>ma27_meminc_factor</td>
<td>Increment factor for workspace size for MA27.</td>
<td>2</td>
</tr>
<tr>
<td>ma27_pivtol</td>
<td>Pivot tolerance for the linear solver MA27.</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma27_pivtolmax</td>
<td>Maximum pivot tolerance for the linear solver MA27.</td>
<td>0.0001</td>
</tr>
<tr>
<td>ma27_skip_inertia_check</td>
<td>Always pretend inertia is correct.</td>
<td>no</td>
</tr>
</tbody>
</table>

### 2.16 Ipopt MA28 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma28_pivtol</td>
<td>Pivot tolerance for linear solver MA28.</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### 2.17 Ipopt MA57 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma57_automatic_scaling</td>
<td>Controls MA57 automatic scaling</td>
<td>no</td>
</tr>
<tr>
<td>ma57_block_size</td>
<td>Controls block size used by Level 3 BLAS in MA57BD</td>
<td>16</td>
</tr>
<tr>
<td>ma57_node_amalgamation</td>
<td>Node amalgamation parameter</td>
<td>16</td>
</tr>
<tr>
<td>ma57_pivot_order</td>
<td>Controls pivot order in MA57</td>
<td>5</td>
</tr>
<tr>
<td>ma57_pivtol</td>
<td>Pivot tolerance for the linear solver MA57.</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma57_pivtolmax</td>
<td>Maximum pivot tolerance for the linear solver MA57.</td>
<td>0.0001</td>
</tr>
<tr>
<td>ma57_pre_alloc</td>
<td>Safety factor for work space memory allocation for the linear solver MA57.</td>
<td>1.05</td>
</tr>
<tr>
<td>ma57_small_pivot_flag</td>
<td>If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.</td>
<td>0</td>
</tr>
</tbody>
</table>

### 2.18 Ipopt MA77 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma77_buffer_lpage</td>
<td>Number of scalars per MA77 buffer page</td>
<td>4096</td>
</tr>
<tr>
<td>ma77_buffer_npage</td>
<td>Number of pages that make up MA77 buffer</td>
<td>1600</td>
</tr>
<tr>
<td>ma77_file_size</td>
<td>Target size of each temporary file for MA77, scalars per type</td>
<td>2097152</td>
</tr>
<tr>
<td>ma77_maxstore</td>
<td>Maximum storage size for MA77 in-core mode</td>
<td>0</td>
</tr>
<tr>
<td>ma77_nemin</td>
<td>Node Amalgamation parameter</td>
<td>8</td>
</tr>
<tr>
<td>ma77_order</td>
<td>Controls type of ordering used by HSL_MA77</td>
<td>metis</td>
</tr>
<tr>
<td>ma77_print_level</td>
<td>Debug printing level for the linear solver MA77</td>
<td>-1</td>
</tr>
<tr>
<td>ma77_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
</tbody>
</table>
2.19 Ipopt MA86 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma86_nemin</td>
<td>Node Amalgamation parameter</td>
<td>32</td>
</tr>
<tr>
<td>ma86_order</td>
<td>Controls type of ordering used by HSL_MA86</td>
<td>auto</td>
</tr>
<tr>
<td>ma86_print_level</td>
<td>Debug printing level for the linear solver MA86</td>
<td>-1</td>
</tr>
<tr>
<td>ma86_scaling</td>
<td>Controls scaling of matrix</td>
<td>mc64</td>
</tr>
<tr>
<td>ma86_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma86_static</td>
<td>Static Pivoting Threshold</td>
<td>0</td>
</tr>
<tr>
<td>ma86_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma86_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

2.20 Ipopt MA97 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma97_nemin</td>
<td>Node Amalgamation parameter</td>
<td>8</td>
</tr>
<tr>
<td>ma97_order</td>
<td>Controls type of ordering used by HSL_MA97</td>
<td>auto</td>
</tr>
<tr>
<td>ma97_print_level</td>
<td>Debug printing level for the linear solver MA97</td>
<td>0</td>
</tr>
<tr>
<td>ma97_scaling</td>
<td>Specifies strategy for scaling in HSL_MA97 linear solver</td>
<td>dynamic</td>
</tr>
<tr>
<td>ma97_scaling1</td>
<td>First scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_scaling2</td>
<td>Second scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_scaling3</td>
<td>Third scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma97_solve_blas3</td>
<td>Controls if blas2 or blas3 routines are used for solve</td>
<td>no</td>
</tr>
<tr>
<td>ma97_switch1</td>
<td>First switch, determine when ma97_scaling1 is enabled.</td>
<td>od_hd_reuse</td>
</tr>
<tr>
<td>ma97_switch2</td>
<td>Second switch, determine when ma97_scaling2 is enabled.</td>
<td>never</td>
</tr>
<tr>
<td>ma97_switch3</td>
<td>Third switch, determine when ma97_scaling3 is enabled.</td>
<td>never</td>
</tr>
<tr>
<td>ma97_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma97_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

2.21 Ipopt Mumps Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mumps_dep_tol</td>
<td>Pivot threshold for detection of linearly dependent constraints in MUMPS.</td>
<td>0</td>
</tr>
<tr>
<td>mumps_mem_percent</td>
<td>Percentage increase in the estimated working space for MUMPS.</td>
<td>1000</td>
</tr>
<tr>
<td>mumps_permuting_scaling</td>
<td>Controls permuting and scaling in MUMPS</td>
<td>7</td>
</tr>
<tr>
<td>mumps_pivot_order</td>
<td>Controls pivot order in MUMPS</td>
<td>7</td>
</tr>
<tr>
<td>mumps_pivot</td>
<td>Pivot tolerance for the linear solver MUMPS.</td>
<td>1e-06</td>
</tr>
<tr>
<td>mumps_pivotmax</td>
<td>Maximum pivot tolerance for the linear solver MUMPS.</td>
<td>0.1</td>
</tr>
</tbody>
</table>
### 2.22 Ipopt NLP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound_relax_factor</td>
<td>Factor for initial relaxation of the bounds.</td>
<td>1e-10</td>
</tr>
<tr>
<td>check_derivatives_for_naninf</td>
<td>Indicates whether it is desired to check for Nan/Inf in derivative matrices</td>
<td>no</td>
</tr>
<tr>
<td>dependency_detection_with_rhs</td>
<td>Indicates if the right hand sides of the constraints should be considered during dependency detection</td>
<td>no</td>
</tr>
<tr>
<td>dependency_detector</td>
<td>Indicates which linear solver should be used to detect linearly dependent equality constraints.</td>
<td>none</td>
</tr>
<tr>
<td>fixed_variable_treatment</td>
<td>Determines how fixed variables should be handled.</td>
<td>make_parameter</td>
</tr>
<tr>
<td>honor_original_bounds</td>
<td>Indicates whether final points should be projected into original bounds.</td>
<td>yes</td>
</tr>
<tr>
<td>jac_c_constant</td>
<td>Indicates whether all equality constraints are linear</td>
<td>no</td>
</tr>
<tr>
<td>jac_d_constant</td>
<td>Indicates whether all inequality constraints are linear</td>
<td>no</td>
</tr>
<tr>
<td>kappa_d</td>
<td>Weight for linear damping term (to handle one-sided bounds).</td>
<td>1e-05</td>
</tr>
<tr>
<td>num_linear_variables</td>
<td>Number of linear variables</td>
<td>0</td>
</tr>
</tbody>
</table>

### 2.23 Ipopt NLP Scaling

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlp_scaling_constr_target_gradient</td>
<td>Target value for constraint function gradient size.</td>
<td>0</td>
</tr>
<tr>
<td>nlp_scaling_max_gradient</td>
<td>Maximum gradient after NLP scaling.</td>
<td>100</td>
</tr>
<tr>
<td>nlp_scaling_method</td>
<td>Select the technique used for scaling the NLP.</td>
<td>gradient-based</td>
</tr>
<tr>
<td>nlp_scaling_min_value</td>
<td>Minimum value of gradient-based scaling values.</td>
<td>1e-08</td>
</tr>
<tr>
<td>nlp_scaling_obj_target_gradient</td>
<td>Target value for objective function gradient size.</td>
<td>0</td>
</tr>
</tbody>
</table>

### 2.24 Ipopt Output

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf_pr_output</td>
<td>Determines what value is printed in the 'inf_pr' output column.</td>
<td>original</td>
</tr>
<tr>
<td>print_eval_error</td>
<td>Switch to enable printing information about function evaluation errors into the GAMS listing file.</td>
<td>yes</td>
</tr>
<tr>
<td>print_frequency_iter</td>
<td>Determines at which iteration frequency the summarizing iteration output line should be printed.</td>
<td>1</td>
</tr>
<tr>
<td>print_frequency_time</td>
<td>Determines at which time frequency the summarizing iteration output line should be printed.</td>
<td>0</td>
</tr>
<tr>
<td>print_info_string</td>
<td>Enables printing of additional info string at end of iteration output.</td>
<td>no</td>
</tr>
<tr>
<td>print_level</td>
<td>Output verbosity level.</td>
<td>5</td>
</tr>
<tr>
<td>print_timing_statistics</td>
<td>Switch to print timing statistics.</td>
<td>no</td>
</tr>
<tr>
<td>replace_bounds</td>
<td>Indicates if all variable bounds should be replaced by inequality constraints</td>
<td>no</td>
</tr>
</tbody>
</table>
### 2.25 Ipopt Pardiso Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pardiso_matching_strategy</td>
<td>Matching strategy to be used by Pardiso</td>
<td>complete+2x2</td>
</tr>
<tr>
<td>pardiso_max_iterative_refinement_steps</td>
<td>Limit on number of iterative refinement steps.</td>
<td>1</td>
</tr>
<tr>
<td>pardiso_msglvl</td>
<td>Pardiso message level</td>
<td>0</td>
</tr>
<tr>
<td>pardiso_order</td>
<td>Controls the fill-in reduction ordering algorithm for the input matrix.</td>
<td>metis</td>
</tr>
<tr>
<td>pardiso_redo_symbolic_fact_only_if_inertia_wrong</td>
<td>Toggle for handling case when elements were perturbed by Pardiso.</td>
<td>no</td>
</tr>
<tr>
<td>pardiso_repeated_perturbation_means_singular</td>
<td>Interpretation of perturbed elements.</td>
<td>no</td>
</tr>
<tr>
<td>pardiso_skip_inertia_check</td>
<td>Always pretend inertia is correct</td>
<td>no</td>
</tr>
</tbody>
</table>

### 2.26 Ipopt Restoration Phase

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound_mult_reset_threshold</td>
<td>Threshold for resetting bound multipliers after the restoration phase.</td>
<td>1000</td>
</tr>
<tr>
<td>constr_mult_reset_threshold</td>
<td>Threshold for resetting equality and inequality multipliers after restoration phase.</td>
<td>0</td>
</tr>
<tr>
<td>evaluate_orig_obj_at_resto_trial</td>
<td>Determines if the original objective function should be evaluated at restoration phase trial points.</td>
<td>yes</td>
</tr>
<tr>
<td>expect_infeasible_problem</td>
<td>Enable heuristics to quickly detect an infeasible problem.</td>
<td>no</td>
</tr>
<tr>
<td>expect_infeasible_problem_ctol</td>
<td>Threshold for disabling ’expect_infeasible_problem’ option.</td>
<td>0.001</td>
</tr>
<tr>
<td>expect_infeasible_problem_ytol</td>
<td>Multiplier threshold for activating ’expect_infeasible_problem’ option.</td>
<td>1e+08</td>
</tr>
<tr>
<td>max_resto_iter</td>
<td>Maximum number of successive iterations in restoration phase.</td>
<td>3000000</td>
</tr>
<tr>
<td>max_soft_resto_iters</td>
<td>Maximum number of iterations performed successively in soft restoration phase.</td>
<td>10</td>
</tr>
<tr>
<td>required_infeasibility_reduction</td>
<td>Required reduction of infeasibility before leaving restoration phase.</td>
<td>0.9</td>
</tr>
<tr>
<td>resto_failure_feasibility_threshold</td>
<td>Threshold for primal infeasibility to declare failure of restoration phase.</td>
<td>0</td>
</tr>
<tr>
<td>resto_penalty_parameter</td>
<td>Penalty parameter in the restoration phase objective function.</td>
<td>1000</td>
</tr>
<tr>
<td>resto_proximity_weight</td>
<td>Weighting factor for the proximity term in restoration phase objective.</td>
<td>1</td>
</tr>
<tr>
<td>soft_resto_perror_reduction_factor</td>
<td>Required reduction in primal-dual error in the soft restoration phase.</td>
<td>0.9999</td>
</tr>
<tr>
<td>start_with_resto</td>
<td>Tells algorithm to switch to restoration phase in first iteration.</td>
<td>no</td>
</tr>
</tbody>
</table>

### 2.27 Ipopt Step Calculation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast_step_computation</td>
<td>Indicates if the linear system should be solved quickly.</td>
<td>no</td>
</tr>
<tr>
<td>first_hessian_perturbation</td>
<td>Size of first x-s perturbation tried.</td>
<td>0.0001</td>
</tr>
<tr>
<td>jacobian_regularization_exponent</td>
<td>Exponent for mu in the regularization for rank-deficient constraint Jacobians.</td>
<td>0.25</td>
</tr>
<tr>
<td>jacobian_regularization_value</td>
<td>Size of the regularization for rank-deficient constraint Jacobians.</td>
<td>1e-08</td>
</tr>
<tr>
<td>max_hessian_perturbation</td>
<td>Maximum value of regularization parameter for handling negative curvature.</td>
<td>1e+20</td>
</tr>
<tr>
<td>max_refinement_steps</td>
<td>Maximum number of iterative refinement steps per linear system solve.</td>
<td>10</td>
</tr>
</tbody>
</table>
### 2.28 Ipopt Warm Start

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>warm_start_bound_frac</td>
<td>same as bound_frac for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_bound_push</td>
<td>same as bound_push for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_init_point</td>
<td>Warm-start for initial point</td>
<td>no</td>
</tr>
<tr>
<td>warm_start_mult_bound_push</td>
<td>same as mult_bound_push for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_mult_init_max</td>
<td>Maximum initial value for the equality multipliers.</td>
<td>1e+06</td>
</tr>
<tr>
<td>warm_start_slack_bound_frac</td>
<td>same as slack_bound_frac for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_slack_bound_push</td>
<td>same as slack_bound_push for the regular initializer.</td>
<td>0.001</td>
</tr>
</tbody>
</table>

### 3 Detailed Options Description

In the following we give a detailed list of options available for Couenne, including those for the underlying Ipopt and Bonmin solvers.

#### 2mir_cuts (integer): Frequency k (in terms of nodes) for generating 2mir_cuts cuts in branch-and-cut.

If k > 0, cuts are generated every k nodes, if -99 < k < 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts.

(default = 0)

#### acceptable_compl_inf_tol (real): 'Acceptance' threshold for the complementarity conditions.

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also acceptable_tol.

(default = 0.01)

#### acceptable_constr_viol_tol (real): 'Acceptance' threshold for the constraint violation.

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also acceptable_tol.

(default = 0.01)

#### acceptable_dual_inf_tol (real): 'Acceptance' threshold for the dual infeasibility.

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also acceptable_tol.
accept_iter (integer): Number of 'acceptable' iterates before triggering termination.

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates,
assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this
heuristic is disabled.

(default = 1e+10)

acceptable_obj_change_tol (real): 'Acceptance' stopping criterion based on objective function change.

If the relative change of the objective function (scaled by Max(1,|f(x)|)) is less than this value, this part of the
acceptable tolerance termination is satisfied; see also acceptable_tol. This is useful for the quasi-Newton option,
which has trouble to bring down the dual infeasibility.

(default = 1e+20)

acceptable_tol (real): 'Acceptable' convergence tolerance (relative).

Determines which (scaled) overall optimality error is considered to be "acceptable." There are two levels of
termination criteria. If the usual "desired" tolerances (see tol, dual_inf_tol etc) are satisfied at an iteration, the
algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters
"acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired
convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the
"desired" level of accuracy.

(default = 1e-06)

accept_after_max_steps (integer): Accept a trial point after maximal this number of steps.

Even if it does not satisfy line search conditions.

(default = -1)

accept_every_trial_step (string): Always accept the first trial step.

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps,
without global convergence guarantees.

(default = no)

  no don't arbitrarily accept the full step
  yes always accept the full step

adaptive_mu_globalization (string): Globalization strategy for the adaptive mu selection mode.

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode
(Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to
decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive").

(default = obj-constr-filter)

  kkt-error nonmonotone decrease of kkt-error
  never-monotone-mode disables globalization
  obj-constr-filter 2-dim filter for objective and constraint violation

adaptive_mu_kkt_error_red_fact (real): Sufficient decrease factor for 'kkt-error' globalization strategy.

For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient
decrease.

Range: [0, 1]

(default = 0.9999)

adaptive_mu_kkt_error_red_iters (integer): Maximum number of iterations requiring sufficient progress.

For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkt_error_red_iters"
iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.
adaptive_mu_kkt_norm_type (string): Norm used for the KKT error in the adaptive mu globalization strategies.

When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this options is also used in the QualityFunctionMuOracle.

(default = 2-norm-squared)

1-norm use the 1-norm (abs sum)
2-norm use 2-norm
2-norm-squared use the 2-norm squared (sum of squares)
max-norm use the infinity norm (max)

adaptive_mu_monotone_init_factor (real): Determines the initial value of the barrier parameter when switching to the monotone mode.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

(default = 0.8)

adaptive_mu_restore_previous_iterate (string): Indicates if the previous iterate should be restored if the monotone mode is entered.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).

(default = no)

no don’t restore accepted iterate
yes restore accepted iterate

add_only_violated_oa (string): Do we add all OA cuts or only the ones violated by current point?

(default = no)

no Add all cuts
yes Add only violated cuts

aggressive_fbbt (string): Aggressive feasibility-based bound tightening (to use with NLP points)

Aggressive FBBT is a version of probing that also allows to reduce the solution set, although it is not as quick as FBBT. It can be applied up to a certain depth of the B&B tree – see “log_num_abt_per_level”. In general, this option is useful but can be switched off if a problem is too large and seems not to benefit from it.

(default = yes)

no

yes

algorithm (string): Choice of the algorithm.

This will preset some of the options of bonmin depending on the algorithm choice.

(default = B-BB)

b-bb simple branch-and-bound algorithm,
b-eqp ecp cuts based branch-and-cut a la FilMINT.
b-hyb hybrid outer approximation based branch-and-cut,
b-ifp Iterated Feasibility Pump for MINLP.
b-oa OA Decomposition algorithm,
b-qg Quesada and Grossmann branch-and-cut algorithm,
allowable_fraction_gap (real): Specify the value of relative gap under which the algorithm stops.

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of the absolute value of the best known solution value.

default = 0.1

allowable_gap (real): Specify the value of absolute gap under which the algorithm stops.

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this.

default = 0

alpha_for_y (string): Method to determine the step size for constraint multipliers.

This option determines how the step size (alpha_y) will be calculated when updating the constraint multipliers.

(default = primal)

acceptor Call LSAcceptor to get step size for y
bound-mult use step size for the bound multipliers (good for LPs)
dual-and-full use the dual step size, and full step if delta_x < alpha_for_y_tol
full take a full step of size one
max use the max of primal and bound multipliers
min use the min of primal and bound multipliers
min-dual-infeas choose step size minimizing new dual infeasibility
primal use primal step size
primal-and-full use the primal step size, and full step if delta_x < alpha_for_y_tol
safer-min-dual-infeas like 'min_dual_infeas', but safeguarded by 'min' and 'max'

alpha_for_y_tol (real): Tolerance for switching to full equality multiplier steps.

This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

(default = 10)

alpha_min_frac (real): Safety factor for the minimal step size (before switching to restoration phase).

This is gamma_alpha in Eqn. (20) in the implementation paper.

Range: [0, 1]

(default = 0.05)

alpha_red_factor (real): Fractional reduction of the trial step size in the backtracking line search.

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: [0, 1]

(default = 0.5)

art_cutoff (real): Artificial cutoff

Default value is infinity.

(default = maxdouble)

art_lower (real): Artificial lower bound

Default value is -COIN_DBL_MAX.

(default = mindouble)

barrier_tol_factor (real): Factor for mu in barrier stop test.

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter
times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. (This is kappa_\text{epsilon} in implementation paper).
(default = 10)

**bb_log_interval (integer):** Interval at which node level output is printed.

Set the interval (in terms of number of nodes) at which a log on node resolutions (consisting of lower and upper bounds) is given.
(default = 100)

**bb_log_level (integer):** specify main branch-and-bound log level.

Set the level of output of the branch-and-bound: 0 - none, 1 - minimal, 2 - normal low, 3 - normal high
Range: [0, 5]
(default = 1)

**boundtightening_print_level (integer):** Output level for bound tightening code in Couenne

Range: [-2, 12]
(default = 0)

**bound_frac (real):** Desired minimum relative distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound\_push"). (This is kappa_2 in Section 3.6 of implementation paper.)
Range: [0, 0.5]
(default = 0.01)

**bound_mult_init_method (string):** Initialization method for bound multipliers

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound\_mult\_init\_val". If "mu-based" is chosen, each value is initialized to the value of "mu\_init" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.

(default = constant)

- constant set all bound multipliers to the value of bound\_mult\_init\_val
- mu-based initialize to mu\_init/x\_slack

**bound_mult_init_val (real):** Initial value for the bound multipliers.

All dual variables corresponding to bound constraints are initialized to this value.
(default = 1)

**bound_mult_reset_threshold (real):** Threshold for resetting bound multipliers after the restoration phase.

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

(default = 1000)

**bound_push (real):** Desired minimum absolute distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound\_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

(default = 0.01)

**bound_relax_factor (real):** Factor for initial relaxation of the bounds.

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then then bounds relaxation is disabled. (See Eqn.(35) in implementation paper.)
(default = 1e-10)

**branching_object (string):** type of branching object for variable selection

(default = var_obj)

- expr_obj use one object for each nonlinear expression
- var_obj use one object for each variable
- vt_obj use Violation Transfer from Tawarmalani and Sahinidis

**branching_print_level (integer):** Output level for braching code in Couenne

Range: [-2, 12]

(default = 0)

**branch_conv_cuts (string):** Apply convexification cuts before branching (for now only within strong branching)

After applying a branching rule and before resolving the subproblem, generate a round of linearization cuts with the new bounds enforced by the rule.

(default = yes)

- no
- yes

**branch_fbbt (string):** Apply bound tightening before branching

After applying a branching rule and before re-solving the subproblem, apply Bound Tightening.

(default = yes)

- no
- yes

**branch_lp_clamp (real):** Defines safe interval percentage for using LP point as a branching point.

Range: [0, 1]

(default = 0.2)

**branch_lp_clamp_cube (real):** Defines safe interval percentage [0,0.5] for using LP point as a branching point.

Range: [0, 0.5]

(default = 0.2)

**branch_lp_clamp_div (real):** Defines safe interval percentage [0,0.5] for using LP point as a branching point.

Range: [0, 0.5]

(default = 0.2)

**branch_lp_clamp_exp (real):** Defines safe interval percentage [0,0.5] for using LP point as a branching point.

Range: [0, 0.5]

(default = 0.2)

**branch_lp_clamp_log (real):** Defines safe interval percentage [0,0.5] for using LP point as a branching point.

Range: [0, 0.5]

(default = 0.2)

**branch_lp_clamp_negpow (real):** Defines safe interval percentage [0,0.5] for using LP point as a branching point.

Range: [0, 0.5]

(default = 0.2)

**branch_lp_clamp_pow (real):** Defines safe interval percentage [0,0.5] for using LP point as a branching point.

Range: [0, 0.5]
(default = 0.2)

**branch_lp_clamp_prod** *(real)*: Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point.

Range: $[0, 0.5]$

(default = 0.2)

**branch_lp_clamp_sqr** *(real)*: Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point.

Range: $[0, 0.5]$

(default = 0.2)

**branch_lp_clamp_trig** *(real)*: Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point.

Range: $[0, 0.5]$

(default = 0.2)

**branch_midpoint_alpha** *(real)*: Defines convex combination of mid point and current LP point: $b = \alpha \cdot x_{lp} + (1-\alpha) \cdot (lb+ub)/2$.

Range: $[0, 1]$

(default = 0.25)

**branch_pt_select** *(string)*: Chooses branching point selection strategy

(default = mid-point)

balanced minimizes max distance from curve to convexification
lp-central LP point if within $[k,1-k]$ of the bound intervals, middle point otherwise (k defined by branch_lp_clamp)
lp-clamped LP point clamped in $[k,1-k]$ of the bound intervals (k defined by lp_clamp)
mid-point convex combination of current point and mid point
min-area minimizes total area of the two convexifications
no-branch do not branch, return null infeasibility; for testing purposes only

**branch_pt_select_cube** *(string)*: Chooses branching point selection strategy for operator cube.

Default is to use the value of branch_pt_select (value common).

(default = common)

balanced
common
lp-central
lp-clamped
mid-point
min-area
no-branch

**branch_pt_select_div** *(string)*: Chooses branching point selection strategy for operator div.

Default is to use the value of branch_pt_select (value common).

(default = common)

balanced
common
lp-central
lp-clamped
mid-point
min-area
no-branch

`branch_pt_select_exp` (string): Chooses branching point selection strategy for operator `exp`.

Default is to use the value of `branch_pt_select` (value `common`).

(default = `common`)
- balanced
- common
- lp-central
- lp-clamped
- mid-point
- min-area
- no-branch

`branch_pt_select_log` (string): Chooses branching point selection strategy for operator `log`.

Default is to use the value of `branch_pt_select` (value `common`).

(default = `common`)
- balanced
- common
- lp-central
- lp-clamped
- mid-point
- min-area
- no-branch

`branch_pt_select_negpow` (string): Chooses branching point selection strategy for operator `negpow`.

Default is to use the value of `branch_pt_select` (value `common`).

(default = `common`)
- balanced
- common
- lp-central
- lp-clamped
- mid-point
- min-area
- no-branch

`branch_pt_select_pow` (string): Chooses branching point selection strategy for operator `pow`.

Default is to use the value of `branch_pt_select` (value `common`).

(default = `common`)
- balanced
- common
- lp-central
- lp-clamped
mid-point
min-area
no-branch

**branch_pt_select_prod** *(string)*: Chooses branching point selection strategy for operator prod.

Default is to use the value of `branch_pt_select` (value `common`).

**(default = common)**
  - balanced
  - common
  - lp-central
  - lp-clamped
  - mid-point
  - min-area
  - no-branch

**branch_pt_select_sqr** *(string)*: Chooses branching point selection strategy for operator sqr.

Default is to use the value of `branch_pt_select` (value `common`).

**(default = common)**
  - balanced
  - common
  - lp-central
  - lp-clamped
  - mid-point
  - min-area
  - no-branch

**branch_pt_select_trig** *(string)*: Chooses branching point selection strategy for operator trig.

Default is to use the value of `branch_pt_select` (value `common`).

**(default = common)**
  - balanced
  - common
  - lp-central
  - lp-clamped
  - mid-point
  - min-area
  - no-branch

**candidate_sort_criterion** *(string)*: Choice of the criterion to choose candidates in strong-branching

**(default = best-ps-cost)**
  - best-ps-cost Sort by decreasing pseudo-cost
  - least-fractional Sort by increasing integer infeasibility
  - most-fractional Sort by decreasing integer infeasibility
  - worst-ps-cost Sort by increasing pseudo-cost

**check_derivatives_for_naninf** *(string)*: Indicates whether it is desired to check for Nan/Inf in derivative matrices
Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an invalid number is detected, the matrix is written to output with print_level corresponding to J_MORE DETAILED; so beware of large output!

(default = no)
no Don’t check (faster).
yes Check Jacobians and Hessian for Nan and Inf.

check_lp (string): Check all LPs through an independent call to OsiClpSolverInterface::initialSolve()
(default = no)
no
yes

clique_cuts (integer): Frequency k (in terms of nodes) for generating clique_cuts cuts in branch-and-cut.
See option 2mir_cuts for the meaning of k.
(default = 0)

coeff_var_threshold (real): Coefficient of variation threshold (for dynamic definition of cutoff_decr).
(default = 0.1)

compl_inf_tol (real): Desired threshold for the complementarity conditions.
Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.
(default = 0.0001)

constraint_violation_norm_type (string): Norm to be used for the constraint violation in the line search.
Determines which norm should be used when the algorithm computes the constraint violation in the line search.
(default = 1-norm)
1-norm use the 1-norm
2-norm use the 2-norm
max-norm use the infinity norm

constr_mult_init_max (real): Maximum allowed least-square guess of constraint multipliers.
Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This options is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero.
(default = 1000)

constr_mult_reset_threshold (real): Threshold for resetting equality and inequality multipliers after restoration phase.
After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.
(default = 0)

constr_viol_tol (real): Desired threshold for the constraint violation.
Absolute tolerance on the constraint violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold.
(default = 0.0001)

cont_var_priority (integer): Priority of continuous variable branching
When branching, this is compared to the priority of integer variables, whose priority is given by int_var_priority, and SOS, whose priority is 10. Higher values mean smaller priority.
(default = 2000)

**convexification_cuts** *(integer)*: Specify the frequency (in terms of nodes) at which couenne ecp cuts are generated.

A frequency of 0 amounts to never solve the NLP relaxation.

(default = 1)

**convexification_points** *(integer)*: Specify the number of points at which to convexify when convexification type is uniform-grid or around-current-point.

(default = 4)

**convexification_type** *(string)*: Determines in which point the linear over/under-estimator are generated

For the lower envelopes of convex functions, this is the number of points where a supporting hyperplane is generated. This only holds for the initial linearization, as all other linearizations only add at most one cut per expression.

(default = current-point-only)

- around-current-point At points around current optimum of relaxation
- current-point-only Only at current optimum of relaxation
- uniform-grid Points chosen in a uniform grid between the bounds of the problem

**convexifying_print_level** *(integer)*: Output level for convexifying code in Couenne

Range: [-2, 12]

(default = 0)

**corrector_compl_avrg_red_fact** *(real)*: Complementarity tolerance factor for accepting corrector step (unsupported!).

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted.

(default = 1)

**corrector_type** *(string)*: The type of corrector steps that should be taken (unsupported!).

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried.

(default = none)

- affine corrector step towards mu=0
- none no corrector
- primal-dual corrector step towards current mu

**cover_cuts** *(integer)*: Frequency k (in terms of nodes) for generating cover_cuts cuts in branch-and-cut.

See option 2mir_cuts for the meaning of k.

(default = 0)

**cutoff** *(real)*: Specify cutoff value.

cutoff should be the value of a feasible solution known by the user (if any). The algorithm will only look for solutions better than cutoff.

Range: [-1e+100, 1e+100]

(default = 1e+100)

**cutoff_decr** *(real)*: Specify cutoff decrement.

Specify the amount by which cutoff is decremented below a new best upper-bound (usually a small positive value but in non-convex problems it may be a negative value).

Range: [-1e+10, 1e+10]

(default = 1e-05)
**delete_redundant** *(string)*: Eliminate redundant variables, which appear in the problem as \( x_k = x_h \)

(default = **yes**)  
- **no**: Keep redundant variables, making the problem a bit larger  
- **yes**: Eliminate redundant variables (the problem will be equivalent, only smaller)

**delta** *(real)*: Multiplier for constraint violation in the switching rule.  
(See Eqn. (19) in the implementation paper.)  
(default = 1)

**dependency_detection_with_rhs** *(string)*: Indicates if the right hand sides of the constraints should be considered during dependency detection  
(default = **no**)  
- **no**: only look at gradients  
- **yes**: also consider right hand side

**dependency_detector** *(string)*: Indicates which linear solver should be used to detect linearly dependent equality constraints.  
The default and available choices depend on how Ipopt has been compiled. This is experimental and does not work well.  
(default = **none**)  
- **ma28**: use MA28  
- **mumps**: use MUMPS  
- **none**: don’t check; no extra work at beginning

**disjcuts_print_level** *(integer)*: Output level for disjunctive cuts in Couenne  
Range: \([-2, 12]\)  
(default = 0)

**disj_active_cols** *(string)*: Only include violated variable bounds in the Cut Generating LP (CGLP).  
This reduces the size of the CGLP, but may produce less efficient cuts.  
(default = **no**)  
- **no**  
- **yes**

**disj_active_rows** *(string)*: Only include violated linear inequalities in the CGLP.  
This reduces the size of the CGLP, but may produce less efficient cuts.  
(default = **no**)  
- **no**  
- **yes**

**disj_cumulative** *(string)*: Add previous disjunctive cut to current CGLP.  
When generating disjunctive cuts on a set of disjunctions 1, 2, ..., k, introduce the cut relative to the previous disjunction \( i-1 \) in the CGLP used for disjunction \( i \). Notice that, although this makes the cut generated more efficient, it increases the rank of the disjunctive cut generated.  
(default = **no**)  
- **no**  
- **yes**

**disj_depth_level** *(integer)*: Depth of the B&B tree when to start decreasing the number of objects that generate disjunctions.  
This has a similar behavior as \( \log_{\text{num}} \text{obbt_per_level} \). A value of -1 means that generation can be done at all
nodes.
(default = 5)

**disj_depth_stop (integer):** Depth of the B&B tree where separation of disjunctive cuts is stopped.

A value of -1 means that generation can be done at all nodes
(default = 20)

**disj_init_number (integer):** Maximum number of disjunction to consider at each iteration.

-1 means no limit.
(default = 10)

**disj_init_perc (real):** The maximum fraction of all disjunctions currently violated by the problem to consider for generating disjunctions.

Range: [0, 1]
(default = 0.5)

**display_stats (string):** display statistics at the end of the run

(default = no)

no

yes

diverging_iterates_tol (real): Threshold for maximal value of primal iterates.

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.
(default = 1e+20)

dual_inf_tol (real): Desired threshold for the dual infeasibility.

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.
(default = 1)

dynamic_def_cutoff_decr (string): Do you want to define the parameter cutoff_decr dynamically?

(default = no)

no

yes

**enable_dynamic_nlp (string):** Enable dynamic linear and quadratic rows addition in nlp

(default = no)

no

yes

**enable_lp IMPLIED_bounds (string):** Enable OsiSolverInterface::tightenBounds () – warning: it has caused some trouble to Couenne

(default = no)

no

yes

**enable_sos (string):** Use Special Ordered Sets (SOS) as indicated in the MINLP model

(default = no)

no

yes
**estimate_select** *(string)*: How the min/max estimates of the subproblems’ bounds are used in strong branching

(default = normal)
- normal as usual in literature
- product use their product

**eta_phi** *(real)*: Relaxation factor in the Armijo condition.

(See Eqn. (20) in the implementation paper)
- Range: \([0, 0.5]\)
- (default = 1e-08)

**evaluate_orig_obj_at_resto_trial** *(string)*: Determines if the original objective function should be evaluated at restoration phase trial points.

Setting this option to "yes" makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.

(default = yes)
- no skip evaluation
- yes evaluate at every trial point

**expect_infeasible_problem** *(string)*: Enable heuristics to quickly detect an infeasible problem.

This options is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.

(default = no)
- no the problem probably be feasible
- yes the problem has a good chance to be infeasible

**expect_infeasible_problem_ctol** *(real)*: Threshold for disabling 'expect_infeasible_problem’ option.

If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem” heuristics in the filter line search are disabled. If the problem is square, this options is set to 0.

(default = 0.001)

**expect_infeasible_problem_ytol** *(real)*: Multiplier threshold for activating 'expect_infeasible_problem’ option.

If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem” is chosen, then the restoration phase is entered.

(default = 1e+08)

**fast_step_computation** *(string)*: Indicates if the linear system should be solved quickly.

If set to yes, the algorithm assumes that the linear system that is solved to obtain the search direction, is solved sufficiently well. In that case, no residuals are computed, and the computation of the search direction is a little faster.

(default = no)
- no Verify solution of linear system by computing residuals.
- yes Trust that linear systems are solved well.

**feasibility_bt** *(string)*: Feasibility-based (cheap) bound tightening (FBBT)
A pre-processing technique to reduce the bounding box, before the generation of linearization cuts. This is a quick and effective way to reduce the solution set, and it is highly recommended to keep it active.

(default = yes)
   no
   yes

**feas.pump_convcuts**(string): Separate MILP-feasible, MINLP-infeasible solution during or after MILP solver.

(default = none)
   external Done after the MILP solver, in a Benders-like fashion
   integrated Done within the MILP solver in a branch-and-cut fashion
   none Just proceed to the NLP
   postcut Do one round of cuts and proceed with NLP

**feas.pump.heuristic**(string): Apply the nonconvex Feasibility Pump

An implementation of the Feasibility Pump for nonconvex MINLPs

(default = no)
   no
   yes

**feas.pump.iter**(integer): Number of iterations in the main Feasibility Pump loop

-1 means no limit

(default = 10)

**feas.pump_level**(integer): Specify the logarithm of the number of feasibility pumps to perform on average for each level of given depth of the tree.

Solve as many nlp’s at the nodes for each level of the tree. Nodes are randomly selected. If for a given level there are less nodes than this number nlp are solved for every nodes. For example if parameter is 8, nlp’s are solved for all node until level 8, then for half the node at level 9, 1/4 at level 10,... Set to -1 to perform at all nodes.

(default = 3)

**feas.pump.milpmethod**(integer): How should the integral solution be constructed?

0: automatic, 1: aggressive heuristics, large node limit, 2: default, node limit, 3: RENS, 4: Objective Feasibility Pump, -1: solve MILP completely

Range: [-1, 4]

(default = -1)

**feas.pump_mult_dist.milp**(real): Weight of the distance in the distance function of the milp problem

0: no weight, 1: full weight

Range: [0, 1]

(default = 0)

**feas.pump_mult_dist.nlp**(real): Weight of the distance in the distance function of the nlp problem

0: no weight, 1: full weight

Range: [0, 1]

(default = 0)

**feas.pump_mult_hess.milp**(real): Weight of the Hessian in the distance function of the milp problem

0: no weight, 1: full weight

Range: [0, 1]
(default = 0)

**feas_pump_mult_hess_nlp (real):** Weight of the Hessian in the distance function of the nlp problem

0: no weight, 1: full weight

Range: [0, 1]

(default = 0)

**feas_pump_mult_objf_milp (real):** Weight of the original objective function in the distance function of the milp problem

0: no weight, 1: full weight

Range: [0, 1]

(default = 0)

**feas_pump_mult_objf_nlp (real):** Weight of the original objective function in the distance function of the nlp problem

0: no weight, 1: full weight

Range: [0, 1]

(default = 0)

**feas_pump_nseprounds (integer):** Number of rounds that separate convexification cuts. Must be at least 1

Range: [1, 100000]

(default = 4)

**feas_pump_poolcomp (integer):** Priority field to compare solutions in FP pool

0: total number of infeasible objects (integer and nonlinear), 1: maximum infeasibility (integer or nonlinear), 2: objective value.

Range: [0, 2]

(default = 0)

**feas_pump_tabumgt (string):** Retrieval of MILP solutions when the one returned is unsatisfactory

(default = pool)

- cut Separate convexification cuts
- none Bail out of feasibility pump
- perturb Randomly perturb unsatisfactory solution
- pool Use a solution pool and replace unsatisfactory solution with Euclidean-closest in pool

**feas_pump_usescip (string):** Should SCIP be used to solve the MILPs?

Note, that SCIP is only available for GAMS users with an academic GAMS license.

(default = yes)

- no Use Cbc’s branch-and-cut to solve the MILP
- yes Use SCIP’s branch-and-cut or heuristics (see feas_pump_milpmethod option) to solve the MILP

**feas_pump_vardist (string):** Distance computed on integer-only or on both types of variables, in different flavors.

(default = integer)

- all Compute the distance using continuous and integer variables
- int-postprocess Use a post-processing fixed-IP LP to determine a closest-point solution
- integer Only compute the distance based on integer coordinates (use post-processing if numerical errors occur)

**feas_tolerance (real):** Tolerance for constraints/auxiliary variables

Default value is 1e-5.
filter_margin_fact (real): Factor determining width of margin for obj-constr-filter adaptive globalization strategy.

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: (new obj) < (filter$obj$) - filter_margin_fact*(new constr-viol) OR (new constr-viol) < (filter constr-viol) - filter_margin_fact*(new constr-viol). For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: [0, 1]
(default = 1e-05)

filter_max_margin (real): Maximum width of margin in obj-constr-filter adaptive globalization strategy.

(default = 1e-05)

filter_reset_trigger (integer): Number of iterations that trigger the filter reset.

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

(default = 5)

first_hessian_perturbation (real): Size of first x-s perturbation tried.

The first value tried for the x-s perturbation in the inertia correction scheme. (This is delta_0 in the implementation paper.)

(default = 0.0001)

first_perc_for_cutoff_decr (real): The percentage used when, the coeff of variance is smaller than the threshold, to compute the cutoff_decr dynamically.

(default = -0.02)

fixed_mu_oracle (string): Oracle for the barrier parameter when switching to fixed mode.

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy".)

(default = average_compl)

average_compl base on current average complementarity
loqo LOQO’s centrality rule
probing Mehrotra’s probing heuristic
quality-function minimize a quality function

fixed_variable_treatment (string): Determines how fixed variables should be handled.

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to "bound.relax.factor"). For both "make_constraints" and "relax_bounds", bound multipliers are computed for the fixed variables.

(default = make_parameter)

make_constraint Add equality constraints fixing variables
make_parameter Remove fixed variable from optimization variables
relax_bounds Relax fixing bound constraints

fixpoint_bt (integer): The frequency (in terms of nodes) at which Fix Point Bound Tightening is performed.

A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne to generate them at every n nodes of the B&B tree. A negative number -n means that generation should be attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.
fixpoint_bt_model (string): Choose whether to add an extended fixpoint LP model or a more compact one.
  (default = compact)
  compact
  extended

flow_covers_cuts (integer): Frequency k (in terms of nodes) for generating flow_covers_cuts cuts in branch-and-cut.
  See option 2mir_cuts for the meaning of k.
  (default = 0)

gamma_phi (real): Relaxation factor in the filter margin for the barrier function.
  (See Eqn. (18a) in the implementation paper.)
  Range: [0, 1]
  (default = 1e-08)

gamma_theta (real): Relaxation factor in the filter margin for the constraint violation.
  (See Eqn. (18b) in the implementation paper.)
  Range: [0, 1]
  (default = 1e-05)

Gomory_cuts (integer): Frequency k (in terms of nodes) for generating Gomory_cuts cuts in branch-and-cut.
  See option 2mir_cuts for the meaning of k.
  (default = 0)

hessian_approximation (string): Indicates what Hessian information is to be used.
  This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.
  (default = exact)
  exact Use second derivatives provided by the NLP.
  limited-memory Perform a limited-memory quasi-Newton approximation

hessian_approximation_space (string): Indicates in which subspace the Hessian information is to be approximated.
  (default = nonlinear-variables)
  all-variables in space of all variables (without slacks)
  nonlinear-variables only in space of nonlinear variables.

honor_original_bounds (string): Indicates whether final points should be projected into original bounds.
  Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization.
  (default = yes)
  no Leave final point unchanged
  yes Project final point back into original bounds

inf_pr_output (string): Determines what value is printed in the 'inf_pr' output column.
  Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice "internal" prints out the constraint violation of this formulation. With "original" the true constraint violation in the original NLP is printed.
  (default = original)
  internal max-norm of violation of internal equality constraints
original maximal constraint violation in original NLP

**integer_tolerance (real):** Set integer tolerance.

  Any number within that value of an integer is considered integer.
  
  (default = 1e-06)

**int_var_priority (integer):** Priority of integer variable branching

  When branching, this is compared to the priority of continuous variables, whose priority is given by cont_var_priority, and SOS, whose priority is 10. Higher values mean smaller priority.
  
  (default = 1000)

**iteration_limit (integer):** Set the cumulative maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound.

  value 0 deactivates option.
  
  (default = maxint)

**iterative_rounding_aggressiveness (integer):** Aggressiveness of the Iterative Rounding heuristic

  Set the aggressiveness of the heuristic; i.e., how many iterations should be run, and with which parameters. The maximum time can be overridden by setting the _time and _time_firstcall options. 0 = non aggressive, 1 = standard (default), 2 = aggressive.

  Range: [0, 2]
  
  (default = 1)

**iterative_rounding_base_lbrhs (integer):** Base rhs of the local branching constraint for Iterative Rounding

  Base rhs for the local branching constraint that defines a neighbourhood of the local incumbent. The base rhs is modified by the algorithm according to variable bounds. This corresponds to k’ in the paper. Default 15.

  (default = 15)

**iterative_rounding_heuristic (string):** Do we use the Iterative Rounding heuristic

  If enabled, a heuristic based on Iterative Rounding is used to find feasible solutions for the problem. The heuristic may take some time, but usually finds good solutions. Recommended if you want good upper bounds and have Cplex. Not recommended if you do not have Cplex.

  (default = no)

  no

  yes

**iterative_rounding_num fir points (integer):** Max number of points rounded at the beginning of Iterative Rounding

  Number of different points (obtained solving a log-barrier problem) that the heuristic will try to round at most, during its execution at the root node (i.e. the F-IR heuristic). Default 5.

  (default = 5)

**iterative_rounding_omega (real):** Omega parameter of the Iterative Rounding heuristic

  Set the omega parameter of the heuristic, which represents a multiplicative factor for the minimum log-barrier parameter of the NLP which is solved to obtain feasible points. This corresponds to \( \omega' \) in the paper. Default 0.2.

  Range: [0, 1]

  (default = 0.2)

**iterative_rounding_time (real):** Specify the maximum time allowed for the Iterative Rounding heuristic

  Maximum CPU time employed by the Iterative Rounding heuristic; if no solution found in this time, failure is reported. This overrides the CPU time set by Aggressiveness if positive.

  (default = -1)
iterative_rounding_time_firstcall (real): Specify the maximum time allowed for the Iterative Rounding heuristic when no feasible solution is known.

Maximum CPU time employed by the Iterative Rounding heuristic when no solution is known; if no solution found in this time, failure is reported. This overrides the CPU time set by Aggressiveness if positive.

(default = -1)
jacobian_regularization_exponent (real): Exponent for \( \mu \) in the regularization for rank-deficient constraint Jacobians.

(This is \( \kappa_c \) in the implementation paper.)

(default = 0.25)
jacobian_regularization_value (real): Size of the regularization for rank-deficient constraint Jacobians.

(This is \( \bar{\delta}_c \) in the implementation paper.)

(default = 1e-08)
jac_c_constant (string): Indicates whether all equality constraints are linear.

Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.

(default = no)

no  Don’t assume that all equality constraints are linear

yes  Assume that equality constraints Jacobian are constant

jac_d_constant (string): Indicates whether all inequality constraints are linear.

Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

(default = no)

no  Don’t assume that all inequality constraints are linear

yes  Assume that equality constraints Jacobian are constant

kappa_d (real): Weight for linear damping term (to handle one-sided bounds).

(see Section 3.7 in implementation paper.)

(default = 1e-05)
kappa_sigma (real): Factor limiting the deviation of dual variables from primal estimates.

If the dual variables deviate from their primal estimates, a correction is performed. (See Eqn. (16) in the implementation paper.) Setting the value to less than 1 disables the correction.

(default = 1e+10)
kappa_soc (real): Factor in the sufficient reduction rule for second order correction.

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. (See Step A-5.9 of Algorithm A in the implementation paper.)

(default = 0.99)
least_square_init_duals (string): Least square initialization of all dual variables.

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn’t know anything about the starting point, or for solving an LP or QP. This overwrites option "bound_mult_init_method".

(default = no)

no  use bound_mult_init_val and least-square equality constraint multipliers

yes  overwrite user-provided point with least-square estimates
least_square_init_primal (string): Least square initialization of the primal variables

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s), to fit the linearized equality and inequality constraints. This might be useful if the user doesn’t know anything about the starting point, or for solving an LP or QP.

(default = no)

   no take user-provided point
   yes overwrite user-provided point with least-square estimates

lift_and_project_cuts (integer): Frequency k (in terms of nodes) for generating lift_and_project_cuts cuts in branch-and-cut.

See option 2mir_cuts for the meaning of k.

(default = 0)

limited_memory_aug_solver (string): Strategy for solving the augmented system for low-rank Hessian.

(default = sherman-morrison)

   extended use an extended augmented system
   sherman-morrison use Sherman-Morrison formula

limited_memory_initialization (string): Initialization strategy for the limited memory quasi-Newton approximation.

Determine how the diagonal Matrix B_{0} as the first term in the limited memory approximation should be computed.

(default = scalar1)

   constant sigma = limited_memory_init_val
   scalar1 sigma = s^{\top}Ty/s^{\top}Ts
   scalar2 sigma = y^{\top}Ty/s^{\top}Ty
   scalar3 arithmetic average of scalar1 and scalar2
   scalar4 geometric average of scalar1 and scalar2

limited_memory_init_val (real): Value for B_{0} in low-rank update.

The starting matrix in the low rank update, B_{0}, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1)

limited_memory_init_val_max (real): Upper bound on value for B_{0} in low-rank update.

The starting matrix in the low rank update, B_{0}, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1e+08)

limited_memory_init_val_min (real): Lower bound on value for B_{0} in low-rank update.

The starting matrix in the low rank update, B_{0}, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1e-08)

limited_memory_max_history (integer): Maximum size of the history for the limited quasi-Newton Hessian approximation.

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.

(default = 6)

limited_memory_max_skipping (integer): Threshold for successive iterations where update is skipped.
If the update is skipped more than this number of successive iterations, we quasi-Newton approximation is reset.
(default = 2)

**limited_memory_special_for_resto (string):** Determines if the quasi-Newton updates should be special during the restoration phase.

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

(default = no)

- no use the same update as in regular iterations
- yes use the a special update during restoration phase

**limited_memory_update_type (string):** Quasi-Newton update formula for the limited memory approximation.

Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

(default = bfgs)

- bfgs BFGS update (with skipping)
- sr1 SR1 (not working well)

**linear_scaling_on_demand (string):** Flag indicating that linear scaling is only done if it seems required.

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.

(default = yes)

- no Always scale the linear system.
- yes Start using linear system scaling if solutions seem not good.

**linear_solver (string):** Linear solver used for step computations.

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, the code must have been compiled with the linear solver you want to choose. Depending on your Ipopt installation, not all options are available.

(default = ma27)

- ma27 use the Harwell routine MA27
- ma57 use the Harwell routine MA57
- ma77 use the Harwell routine HSL_MA77
- ma86 use the Harwell routine HSL_MA86
- ma97 use the Harwell routine HSL_MA97
- mumps use MUMPS package
- pardiso use the Pardiso package

**linear_system_scaling (string):** Method for scaling the linear system.

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. By default, MC19 is only used if MA27 or MA57 are selected as linear solvers. This value is only available if Ipopt has been compiled with MC19.

(default = mc19)

- mc19 use the Harwell routine MC19
- none no scaling will be performed
slack-based use the slack values

**line_search_method** *(string)*: Globalization method used in backtracking line search

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.

(default = filter)

- **cg-penalty**: Chen-Goldfarb penalty function
- **filter**: Filter method
- **penalty**: Standard penalty function

**local_branching_heuristic** *(string)*: Apply local branching heuristic

A local-branching heuristic based is used to find feasible solutions.

(default = no)

- **no**
- **yes**

**local_optimization_heuristic** *(string)*: Search for local solutions of MINLPs

If enabled, a heuristic based on Ipopt is used to find feasible solutions for the problem. It is highly recommended that this option is left enabled, as it would be difficult to find feasible solutions otherwise.

(default = yes)

- **no**
- **yes**

**log_num_abt_per_level** *(integer)*: Specify the frequency (in terms of nodes) for aggressive bound tightening.

If -1, apply at every node (expensive!). If 0, apply at root node only. If k ≥ 0, apply with probability $2^k(k - \text{level})$, level being the current depth of the B&B tree.

(default = 2)

**log_num_local_optimization_per_level** *(integer)*: Specify the logarithm of the number of local optimizations to perform on average for each level of given depth of the tree.

Solve as many nlp’s at the nodes for each level of the tree. Nodes are randomly selected. If for a given level there are less nodes than this number nlp are solved for every nodes. For example if parameter is 8, nlp’s are solved for all node until level 8, then for half the node at level 9, 1/4 at level 10.... Value -1 specify to perform at all nodes.

(default = 2)

**log_num_obbt_per_level** *(integer)*: Specify the frequency (in terms of nodes) for optimality-based bound tightening.

If -1, apply at every node (expensive!). If 0, apply at root node only. If k ≥ 0, apply with probability $2^k(k - \text{level})$, level being the current depth of the B&B tree.

(default = 1)

**lp_log_level** *(integer)*: specify LP log level.

Set the level of output of the linear programming sub-solver in B-Hyb or B-QG : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high, 4 - verbose

Range: [0, 4]

(default = 0)

**lp_solver** *(string)*: Linear Programming solver for the linearization

(default = clp)

- **clp**: Use the COIN-OR Open Source solver CLP
- **cplex**: Use the commercial solver Cplex (license is needed)
gurobi Use the commercial solver Gurobi (license is needed)
soplex Use the freely available Soplex
xpress-mp Use the commercial solver Xpress MP (license is needed)

**ma27.ignore_singularity (string):** Enables MA27’s ability to solve a linear system even if the matrix is singular.
Setting this option to “yes” means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt’s heuristic of small perturbation of the lower diagonal of the KKT matrix.

(default = no)
- no Don’t have MA27 solve singular systems
- yes Have MA27 solve singular systems

**ma27.la_init_factor (real):** Real workspace memory for MA27.
The initial real workspace memory = la_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

(default = 5)

**ma27.liw_init_factor (real):** Integer workspace memory for MA27.
The initial integer workspace memory = liw_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

(default = 5)

**ma27.meminc_factor (real):** Increment factor for workspace size for MA27.
If the integer or real workspace is not large enough, Ipopt will increase its size by this factor. This option is only available if Ipopt has been compiled with MA27.

(default = 2)

**ma27.pivtol (real):** Pivot tolerance for the linear solver MA27.
A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA27.

Range: [0, 1]

(default = 1e−08)

**ma27.pivtolmax (real):** Maximum pivot tolerance for the linear solver MA27.
Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA27.

Range: [0, 1]

(default = 0.0001)

**ma27.skip_inertia_check (string):** Always pretend inertia is correct.
Setting this option to “yes” essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

(default = no)
- no check inertia
- yes skip inertia check

**ma28.pivtol (real):** Pivot tolerance for linear solver MA28.
This is used when MA28 tries to find the dependent constraints.

Range: [0, 1]
**ma57**

**ma57.automatic.scaling** *(string)*: Controls MA57 automatic scaling

This option controls the internal scaling option of MA57. For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

*(default = no)*

- no Do not scale the linear system matrix
- yes Scale the linear system matrix

**ma57.block.size** *(integer)*: Controls block size used by Level 3 BLAS in MA57BD

This is ICNTL(11) in MA57.

*(default = 16)*

**ma57.node.amalgamation** *(integer)*: Node amalgamation parameter

This is ICNTL(12) in MA57.

*(default = 16)*

**ma57.pivot.order** *(integer)*: Controls pivot order in MA57

This is ICNTL(6) in MA57.

- Range: [0, 5]
- *(default = 5)*

**ma57.pivot** *(real)*: Pivot tolerance for the linear solver MA57.

- A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA57.
- Range: [0, 1]
- *(default = 1e-08)*

**ma57.pivotmax** *(real)*: Maximum pivot tolerance for the linear solver MA57.

- Ipopt may increase pivot as high as ma57.pivotmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.
- Range: [0, 1]
- *(default = 0.0001)*

**ma57.pre.alloc** *(real)*: Safety factor for work space memory allocation for the linear solver MA57.

- If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice. This option is only available if Ipopt has been compiled with MA57.
- *(default = 1.05)*

**ma57.small.pivot.flag** *(integer)*: If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.

- This is ICNTL(16) in MA57.
- Range: [0, 1]
- *(default = 0)*

**ma77.buffer.lpage** *(integer)*: Number of scalars per MA77 buffer page

- Number of scalars per an in-core buffer in the out-of-core solver MA77. Must be at most ma77.file.size.
- *(default = 4096)*

**ma77.buffer.npage** *(integer)*: Number of pages that make up MA77 buffer
Number of pages of size buffer_filepage that exist in-core for the out-of-core solver MA77.
(defualt = 1600)

**ma77_file_size (integer)**: Target size of each temporary file for MA77, scalars per type

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.
(defualt = 2097152)

**ma77_maxstore (integer)**: Maximum storage size for MA77 in-core mode

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.
(defualt = 0)

**ma77_nemin (integer)**: Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than ma77_nemin variables.
(defualt = 8)

**ma77_order (string)**: Controls type of ordering used by HSL MA77

This option controls ordering for the solver HSL_MA77.
(defualt = metis)

- _amd_ Use the HSL_MC68 approximate minimum degree algorithm
- _metis_ Use the MeTiS nested dissection algorithm (if available)

**ma77_print_level (integer)**: Debug printing level for the linear solver MA77
(defualt = -1)

**ma77_small (real)**: Zero Pivot Threshold

Any pivot less than ma77_small is treated as zero.
(defualt = 1e-20)

**ma77_static (real)**: Static Pivoting Threshold

See MA77 documentation. Either ma77_static=0.0 or ma77_static>ma77_small. ma77_static=0.0 disables static pivoting.
(defualt = 0)

**ma77_u (real)**: Pivoting Threshold

See MA77 documentation.
Range: [0, 0.5]
(defualt = 1e-08)

**ma77_umax (real)**: Maximum Pivoting Threshold

Maximum value to which u will be increased to improve quality.
Range: [0, 0.5]
(defualt = 0.0001)

**ma86_nemin (integer)**: Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than ma86_nemin variables.
(defualt = 32)

**ma86_order (string)**: Controls type of ordering used by HSL MA86

This option controls ordering for the solver HSL_MA86.
(defualt = auto)
amd Use the HSL_MC68 approximate minimum degree algorithm
auto Try both AMD and MeTiS, pick best
metis Use the MeTiS nested dissection algorithm (if available)

**ma86_print_level** (*integer*): Debug printing level for the linear solver MA86
(default = -1)

**ma86scaling** (*string*): Controls scaling of matrix
This option controls scaling for the solver HSL_MA86.

(default = mc64)
- mc64 Scale linear system matrix using MC64
- mc77 Scale linear system matrix using MC77 [1,3,0]
- none Do not scale the linear system matrix

**ma86_small** (*real*): Zero Pivot Threshold
Any pivot less than ma86_small is treated as zero.
(default = 1e-20)

**ma86_static** (*real*): Static Pivoting Threshold
See MA86 documentation. Either ma86_static=0.0 or ma86_static>ma86_small. ma86_static=0.0 disables static pivoting.
(default = 0)

**ma86_u** (*real*): Pivoting Threshold
See MA86 documentation.
Range: [0, 0.5]
(default = 1e-08)

**ma86_umax** (*real*): Maximum Pivoting Threshold
Maximum value to which u will be increased to improve quality.
Range: [0, 0.5]
(default = 0.0001)

**ma97_nemin** (*integer*): Node Amalgamation parameter
Two nodes in elimination tree are merged if result has fewer than ma97_nemin variables.
(default = 8)

**ma97_order** (*string*): Controls type of ordering used by HSL_MA97
(default = auto)

- amd Use the HSL_MC68 approximate minimum degree algorithm
- auto Use HSL_MA97 heuristic to guess best of AMD and METIS
- best Try both AMD and MeTiS, pick best
- matched-amd Use the HSL_MC80 matching based ordering with AMD
- matched-auto Use the HSL_MC80 matching with heuristic choice of AMD or METIS
- matched-metis Use the HSL_MC80 matching based ordering with METIS
- metis Use the MeTiS nested dissection algorithm

**ma97_print_level** (*integer*): Debug printing level for the linear solver MA97
(default = 0)
**ma97_scaling (string):** Specifies strategy for scaling in HSL_MA97 linear solver

(default = dynamic)

- **dynamic**: Dynamically select scaling according to rules specified by ma97_scalingX and ma97_switchX options.
- **mc30**: Scale all linear system matrices using MC30
- **mc64**: Scale all linear system matrices using MC64
- **mc77**: Scale all linear system matrices using MC77 [1,3,0]
- **none**: Do not scale the linear system matrix

**ma97_scaling1 (string):** First scaling.

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch1. If ma97_switch2 is triggered it is disabled.

(default = mc64)

- **mc30**: Scale linear system matrix using MC30
- **mc64**: Scale linear system matrix using MC64
- **mc77**: Scale linear system matrix using MC77 [1,3,0]
- **none**: No scaling

**ma97_scaling2 (string):** Second scaling.

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch2. If ma97_switch3 is triggered it is disabled.

(default = mc64)

- **mc30**: Scale linear system matrix using MC30
- **mc64**: Scale linear system matrix using MC64
- **mc77**: Scale linear system matrix using MC77 [1,3,0]
- **none**: No scaling

**ma97_scaling3 (string):** Third scaling.

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch3.

(default = mc64)

- **mc30**: Scale linear system matrix using MC30
- **mc64**: Scale linear system matrix using MC64
- **mc77**: Scale linear system matrix using MC77 [1,3,0]
- **none**: No scaling

**ma97_small (real):** Zero Pivot Threshold

Any pivot less than ma97_small is treated as zero.

(default = 1e−20)

**ma97_solve_blas3 (string):** Controls if blas2 or blas3 routines are used for solve

(default = no)

- **no**: Use BLAS2 (faster, some implementations bit incompatible)
- **yes**: Use BLAS3 (slower)

**ma97_switch1 (string):** First switch, determine when ma97_scaling1 is enabled.

If ma97_scaling=dynamic, ma97_scaling1 is enabled according to this condition. If ma97_switch2 occurs this option is henceforth ignored.
Couenne 699

(default = od_hd_reuse)

at_start Scaling to be used from the very start.
at_start_reuse Scaling to be used on first iteration, then reused thereafter.
high_delay Scaling to be used after more than 0.05*n delays are present
high_delay_reuse Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
never Scaling is never enabled.
od_hd Combination of on_demand and high_delay
od_hd_reuse Combination of on_demand_reuse and high_delay_reuse
on_demand Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse As on_demand, but reuse scaling from previous itr

ma97_switch2 (string): Second switch, determine when ma97_scaling2 is enabled.

If ma97_scaling=dynamic, ma97_scaling2 is enabled according to this condition. If ma97_switch3 occurs this option is henceforth ignored.
(default = never)

at_start Scaling to be used from the very start.
at_start_reuse Scaling to be used on first iteration, then reused thereafter.
high_delay Scaling to be used after more than 0.05*n delays are present
high_delay_reuse Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
never Scaling is never enabled.
od_hd Combination of on_demand and high_delay
od_hd_reuse Combination of on_demand_reuse and high_delay_reuse
on_demand Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse As on_demand, but reuse scaling from previous itr

ma97_switch3 (string): Third switch, determine when ma97_scaling3 is enabled.

If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.
(default = never)

at_start Scaling to be used from the very start.
at_start_reuse Scaling to be used on first iteration, then reused thereafter.
high_delay Scaling to be used after more than 0.05*n delays are present
high_delay_reuse Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
never Scaling is never enabled.
od_hd Combination of on_demand and high_delay
od_hd_reuse Combination of on_demand_reuse and high_delay_reuse
on_demand Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse As on_demand, but reuse scaling from previous itr

ma97_u (real): Pivoting Threshold
See MA97 documentation.
Range: [0, 0.5]
(default = 1e-08)

**ma97\_umax (real)**: Maximum Pivoting Threshold
See MA97 documentation.
Range: [0, 0.5]
(default = 0.0001)

**maxmin\_crit\_have\_sol (real)**: Weight towards minimum in of lower and upper branching estimates when a solution has been found.
Range: [0, 1]
(default = 0.1)

**maxmin\_crit\_no\_sol (real)**: Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.
Range: [0, 1]
(default = 0.7)

**max\_consecutive\_failures (integer)**: (temporarily removed) Number of consecutive unsolved problems before aborting a branch of the tree.
When \( n > 0 \), continue exploring a branch of the tree until \( n \) consecutive problems in the branch are unsolved (we call unsolved a problem for which Ipopt can not guarantee optimality within the specified tolerances).
(default = 10)

**max\_consecutive\_infeasible (integer)**: Number of consecutive infeasible subproblems before aborting a branch.
Will continue exploring a branch of the tree until "max\_consecutive\_infeasible" consecutive problems are locally infeasible by the NLP sub-solver.
(default = 0)

**max\_cpu\_time (real)**: Maximum number of CPU seconds.
A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding error message.
(default = 1e+06)

**max\_fbbt\_iter (integer)**: Number of FBBT iterations before stopping even with tightened bounds.
Set to -1 to impose no upper limit
(default = 3)

**max\_filter\_resets (integer)**: Maximal allowed number of filter resets
A positive number enables a heuristic that resets the filter, whenever in more than "filter\_reset\_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place.
(default = 5)

**max\_hessian\_perturbation (real)**: Maximum value of regularization parameter for handling negative curvature.
In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. (This is delta\(_w^+\)max in the implementation paper.)
max_iter (integer): Maximum number of iterations.
    The algorithm terminates with an error message if the number of iterations exceeded this number.
    (default = 3000)

max_random_point_radius (real): Set max value r for coordinate of a random point.
    When picking a random point, coordinate i will be in the interval [\min(\max(l,-r),u-r), \max(\min(u,r),l+r)] (where l is the lower bound for the variable and u is its upper bound)
    (default = 100000)

max_refinement_steps (integer): Maximum number of iterative refinement steps per linear system solve.
    Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option
determines the maximum number of iterative refinement steps.
    (default = 10)

max_resto_iter (integer): Maximum number of successive iterations in restoration phase.
    The algorithm terminates with an error message if the number of iterations successively taken in the restoration
phase exceeds this number.
    (default = 3000000)

max_soc (integer): Maximum number of second order correction trial steps at each iteration.
    Choosing 0 disables the second order corrections. (This is \(p^\omega\{\max\) of Step A-5.9 of Algorithm A in the
implementation paper.)
    (default = 4)

max_soft_resto_iters (integer): Maximum number of iterations performed successively in soft restoration phase.
    If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase
is called.
    (default = 10)

mehrotra_algorithm (string): Indicates if we want to do Mehrotra’s algorithm.
    If set to yes, Ipopt runs as Mehrotra’s predictor-corrector algorithm. This works usually very well for LPs and
convex QPs. This automatically disables the line search, and chooses the (unglobalized) adaptive mu strategy
with the “probing” oracle, and uses “corrector_type=affine” without any safeguards; you should not set any of
those options explicitly in addition. Also, unless otherwise specified, the values of “bound_push”, “bound_frac”,
and “bound_mult_init_val” are set more aggressive, and sets “alpha_for_y=bound_mult”.
    (default = no)
        no Do the usual Ipopt algorithm.
        yes Do Mehrotra’s predictor-corrector algorithm.

milp_solver (string): Choose the subsolver to solve MILP sub-problems in OA decompositions.
    To use Cplex, a valid license is required.
    (default = Cbc_D)
        cbc_d Coin Branch and Cut with its default
        cbc_par Coin Branch and Cut with passed parameters
        cplex Ilog Cplex

milp_strategy (string): Choose a strategy for MILPs.
    (default = find_good_sol)
        find_good_sol Stop sub milps when a solution improving the incumbent is found
solve_to_optimality Solve MILPs to optimality

minlp_disj_cuts (integer): The frequency (in terms of nodes) at which Couenne disjunctive cuts are generated.
A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne
to generate them at every n nodes of the B&B tree. A negative number -n means that generation should be
attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.
(default = 0)

min_hessian_perturbation (real): Smallest perturbation of the Hessian block.
The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation
is necessary. (This is delta_w^\min in implementation paper.)
(default = 1e-20)

min_number_strong_branch (integer): Sets minimum number of variables for strong branching (overriding trust)
(default = 0)

min_refinement_steps (integer): Minimum number of iterative refinement steps per linear system solve.
Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option
determines the minimum number of iterative refinements (i.e. at least “min_refinement_steps” iterative refinement
steps are enforced per right hand side.)
(default = 1)

mir_cuts (integer): Frequency k (in terms of nodes) for generating mir_cuts cuts in branch-and-cut.
See option 2mir_cuts for the meaning of k.
(default = 0)

multilinear_separation (string): Separation for multilinear terms
Type of separation for multilinear terms where the dependent variable is also bounded
(default = tight)
   none No separation – just use the four McCormick inequalities
   simple Use one considering lower curve only
   tight Use one considering both curves pi(x) = l_{k+1} and pi(x) = u_{k+1}

mumps_dep_tol (real): Pivot threshold for detection of linearly dependent constraints in MUMPS.
When MUMPS is used to determine linearly dependent constraints, this is determines the threshold for a pivot to
be considered zero. This is CNTL(3) in MUMPS.
(default = 0)

mumps_mem_percent (integer): Percentage increase in the estimated working space for MUMPS.
In MUMPS when significant extra fill-in is caused by numerical pivoting, larger values of mumps_mem_percent
may help use the workspace more efficiently. On the other hand, if memory requirement are too large at the very
beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory
requirements.
(default = 1000)

mumps_permuting_scaling (integer): Controls permuting and scaling in MUMPS
This is ICNTL(6) in MUMPS.
Range: [0, 7]
(default = 7)

mumps_pivot_order (integer): Controls pivot order in MUMPS
This is ICNTL(7) in MUMPS.
**mumps_pivot (real):** Pivot tolerance for the linear solver MUMPS.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MUMPS.

- Range: [0, 1]
- (default = 1e-06)

**mumps_pivotmax (real):** Maximum pivot tolerance for the linear solver MUMPS.

Ipopt may increase pivot as high as pivotmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MUMPS.

- Range: [0, 1]
- (default = 0.1)

**mumps_scaling (integer):** Controls scaling in MUMPS.

This is ICNTL(8) in MUMPS.

- Range: [-2, 77]
- (default = 77)

**mu_allow_fast_monotone_decrease (string):** Allow skipping of barrier problem if barrier test is already met.

- If set to "no", the algorithm enforces at least one iteration per barrier problem, even if the barrier test is already met for the updated barrier parameter.

- (default = yes)
  - no: Take at least one iteration per barrier problem
  - yes: Allow fast decrease of mu if barrier test it met

**mu_init (real):** Initial value for the barrier parameter.

This option determines the initial value for the barrier parameter (mu). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if “mu_strategy” is chosen as “monotone”)

- (default = 0.1)

**mu_linear_decrease_factor (real):** Determines linear decrease rate of barrier parameter.

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of mu * "mu_linear_decrease_factor” and mu “superlinear_decrease_power”. (This is kappa_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

- Range: [0, 1]
- (default = 0.2)

**mu_max (real):** Maximum value for barrier parameter.

This option specifies an upper bound on the barrier parameter in the adaptive mu selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option “mu_strategy” is chosen as “adaptive”.)

- (default = 100000)

**mu_max_fact (real):** Factor for initialization of maximum value for barrier parameter.

This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option “mu_strategy” is chosen as “adaptive”).

- (default = 1000)

**mu_min (real):** Minimum value for barrier parameter.
This option specifies the lower bound on the barrier parameter in the adaptive mu selection mode. By default, it is set to the minimum of $1e^{-11}$ and $\min(\text{"tol"},\text{"compl_inf_tol"})/(\text{"barrier_tol_factor"}+1)$, which should be a reasonable value. (Only used if option "mu_strategy" is chosen as "adaptive".)

(default = $1e^{-11}$)

**mu_oracle (string):** Oracle for a new barrier parameter in the adaptive strategy.

Determines how a new barrier parameter is computed in each "free-mode" iteration of the adaptive barrier parameter strategy. (Only considered if "adaptive" is selected for option "mu_strategy").

(default = quality-function)

- logq LOQO’s centrality rule
- probing Mehrotra’s probing heuristic
- quality-function minimize a quality function

**mu_strategy (string):** Update strategy for barrier parameter.

Determines which barrier parameter update strategy is to be used.

(default = monotone)

- adaptive use the adaptive update strategy
- monotone use the monotone (Fiacco-McCormick) strategy

**mu_superlinear_decrease_power (real):** Determines superlinear decrease rate of barrier parameter.

For the Fiacco-McCormick update procedure the new barrier parameter $\mu$ is obtained by taking the minimum of $\mu^{\ast \text{"mu_linear_decrease_factor"}}$ and $\mu^{\omega \text{"superlinear_decrease_power"}}$. (This is $\theta_{\mu}$ in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: $[1, 2]$

(default = 1.5)

**mu_target (real):** Desired value of complementarity.

Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of "mu_target" specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

(default = 0)

**neg_curv_test_tol (real):** Tolerance for heuristic to ignore wrong inertia.

If positive, incorrect inertia in the augmented system is ignored, and we test if the direction is a direction of positive curvature. This tolerance determines when the direction is considered to be sufficiently positive.

(default = 0)

**nlpheur_print_level (integer):** Output level for NLP heuristic in Couenne

Range: $[-2, 12]$

(default = 0)

**nlp_failure_behavior (string):** Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).

If set to "fathom", the algorithm will fathom the node when Ipopt fails to find a solution to the nlp at that node within the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal.

(default = stop)

- fathom Continue when failure happens.
stop Stop when failure happens.

**nlp_log_at_root** *(integer)*: specify a different log level for root relaxation.

- Range: [0, 12]
- (default = 0)

**nlp_log_level** *(integer)*: specify NLP solver interface log level (independent from ipopt print_level).

- Set the level of output of the OsTIMINLPInterface: 0 - none, 1 - normal, 2 - verbose
- Range: [0, 2]
- (default = 1)

**nlp_scaling_constr_target_gradient** *(real)*: Target value for constraint function gradient size.

- If a positive number is chosen, the scaling factor the constraint functions is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the constraint functions.
- (default = 0)

**nlp_scaling_max_gradient** *(real)*: Maximum gradient after NLP scaling.

- This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is $g_{\text{max}}$ in Section 3.8 of the implementation paper.) Note: This option is only used if ”nlp_scaling_method” is chosen as "gradient-based".
- (default = 100)

**nlp_scaling_method** *(string)*: Select the technique used for scaling the NLP.

- Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP. If you are using AMPL, they can be specified through suffixes ("scaling_factor").
- (default = gradient-based)
  - equilibration-based scale the problem so that first derivatives are of order 1 at random points (only available with MC19)
  - gradient-based scale the problem so the maximum gradient at the starting point is scaling_max_gradient
  - none no problem scaling will be performed

**nlp_scaling_min_value** *(real)*: Minimum value of gradient-based scaling values.

- This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if ”nlp_scaling_method” is chosen as "gradient-based".
- (default = 1e-08)

**nlp_scaling_obj_target_gradient** *(real)*: Target value for objective function gradient size.

- If a positive number is chosen, the scaling factor the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the objective function.
- (default = 0)

**node_comparison** *(string)*: Choose the node selection strategy.

- Choose the strategy for selecting the next node to be processed.
- (default = best-bound)
  - best-bound choose node with the smallest bound,
best-guess choose node with smallest guessed integer solution
breadth-first Perform breadth first search,
depth-first Perform depth first search,
dynamic Cbc dynamic strategy (starts with a depth first search and turn to best bound after 3 integer feasible solutions have been found).

node_limit (integer): Set the maximum number of nodes explored in the branch-and-bound search.
  (default = maxint)

number_before_trust (integer): Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching.
  A value of 0 disables pseudo costs.
  (default = 8)

number_before_trust_list (integer): Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list.
  The default value is that of "number_before_trust"
  (default = 0)

number_look_ahead (integer): Sets limit of look-ahead strong-branching trials
  (default = 0)

number_strong_branch (integer): Choose the maximum number of variables considered for strong branching.
  Set the number of variables on which to do strong branching.
  (default = 20)

number_strong_branch_root (integer): Maximum number of variables considered for strong branching in root node.
  (default = maxint)

num_cut_passes (integer): Set the maximum number of cut passes at regular nodes of the branch-and-cut.
  (default = 1)

num_cut_passes_at_root (integer): Set the maximum number of cut passes at regular nodes of the branch-and-cut.
  (default = 20)

num_iterations_suspect (integer): Number of iterations over which a node is considered 'suspect' (for debugging purposes only, see detailed documentation).
  When the number of iterations to solve a node is above this number, the subproblem at this node is considered to be suspect and it will be written into a file (set to -1 to deactivate this).
  (default = -1)

num_linear_variables (integer): Number of linear variables
  When the Hessian is approximated, it is assumed that the first num_linear_variables variables are linear. The Hessian is then not approximated in this space. If the get_number_of_nonlinear_variables method in the TNLP is implemented, this option is ignored.
  (default = 0)

num_resolve_at_infeasibles (integer): Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point.
  The algorithm will solve all the infeasible nodes with k different random starting points and will keep the best local optimum found.
  (default = 0)

num_resolve_at_node (integer): Number k of tries to resolve a node (other than the root) of the tree with different starting point.
The algorithm will solve all the nodes with $k$ different random starting points and will keep the best local optimum found.

(default = 0)

**num_resolve_at_root** (integer): Number $k$ of tries to resolve the root node with different starting points.

The algorithm will solve the root node with $k$ random starting points and will keep the best local optimum found.

(default = 0)

**num_retry_unsolved_random_point** (integer): Number $k$ of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).

When Ipopt fails to solve a continuous NLP sub-problem, if $k > 0$, the algorithm will try again to solve the failed NLP with $k$ new randomly chosen starting points or until the problem is solved with success.

(default = 0)

**nu_inc** (real): Increment of the penalty parameter.

(default = 0.0001)

**nu_init** (real): Initial value of the penalty parameter.

(default = 1e-06)

**oa_cuts_log_level** (integer): level of log when generating OA cuts.

0: outputs nothing, 1: when a cut is generated, its violation and index of row from which it originates, 2: always output violation of the cut. 3: output generated cuts incidence vectors.

(default = 0)

**oa_cuts_scope** (string): Specify if OA cuts added are to be set globally or locally valid

(default = global)

   global Cuts are treated as globally valid
   local Cuts are treated as locally valid

**oa_rhs_relax** (real): Value by which to relax OA cut

RHS of OA constraints will be relaxed by this amount times the absolute value of the initial rhs if it is $\geq 1$ (otherwise by this amount).

(default = 1e-08)

**obj_max_inc** (real): Determines the upper bound on the acceptable increase of barrier objective function.

Trial points are rejected if they lead to an increase in the barrier objective function by more than $\text{obj\_max\_inc}$ orders of magnitude.

(default = 5)

**optimality(bt** (string): Optimality-based (expensive) bound tightening (OBBT)

This is another bound reduction technique aiming at reducing the solution set by looking at the initial LP relaxation. This technique is computationally expensive, and should be used only when necessary.

(default = yes)

   no
   yes

**orbital_branching** (string): detect symmetries and apply orbital branching

(default = no)

   no
   yes
output_level (integer): Output level

Range: [-2, 12]
(default = 0)

pardiso_matching_strategy (string): Matching strategy to be used by Pardiso

This is IPAR(13) in Pardiso manual.
(default = complete+2x2)
  complete Match complete (IPAR(13)=1)
  complete+2x2 Match complete+2x2 (IPAR(13)=2)
  constraints Match constraints (IPAR(13)=3)

pardiso_max_iterative_refinement_steps (integer): Limit on number of iterative refinement steps.

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

(default = 1)

pardiso_msglvl (integer): Pardiso message level

This determines the amount of analysis output from the Pardiso solver. This is MSGVL in the Pardiso manual.

(default = 0)

pardiso_order (string): Controls the fill-in reduction ordering algorithm for the input matrix.

(default = metis)
  amd minimum degree algorithm
  metis MeTiS nested dissection algorithm
  one undocumented
  pmetis parallel (OpenMP) version of MeTiS nested dissection algorithm

pardiso_redo_symbolic_fact_only_if_inertia_wrong (string): Toggle for handling case when elements were perturbed by Pardiso.

(default = no)
  no Always redo symbolic factorization when elements were perturbed
  yes Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

pardiso_repeated_perturbation_means_singular (string): Interpretation of perturbed elements.

(default = no)
  no Don’t assume that matrix is singular if elements were perturbed after recent symbolic factorization
  yes Assume that matrix is singular if elements were perturbed after recent symbolic factorization

pardiso_skip_inertia_check (string): Always pretend inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

(default = no)
  no check inertia
  yes skip inertia check

perturb_always_cd (string): Active permanent perturbation of constraint linearization.
This option makes the delta_c and delta_d perturbation be used for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.
(default = no)

- no perturbation only used when required
- yes always use perturbation

**perturb_dec_fact** *(real)*: Decrease factor for x-s perturbation.

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. (This is kappa_w^- in the implementation paper.)

Range: [0, 1]
(default = 0.333333)

**perturb_inc_fact** *(real)*: Increase factor for x-s perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. (This is kappa_w^+ in the implementation paper.)

(default = 8)

**perturb_inc_fact_first** *(real)*: Increase factor for x-s perturbation for very first perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for for the first perturbation than that used for the remaining perturbations. (This is bar_kappa_w^+ in the implementation paper.)

(default = 100)

**print_eval_error** *(string)*: Switch to enable printing information about function evaluation errors into the GAMS listing file.

(default = yes)

- no
- yes

**print_frequency_iter** *(integer)*: Determines at which iteration frequency the summarizing iteration output line should be printed.

Summarizing iteration output is printed every print_frequency_iter iterations, if at least print_frequency_time seconds have passed since last output.

(default = 1)

**print_frequency_time** *(real)*: Determines at which time frequency the summarizing iteration output line should be printed.

Summarizing iteration output is printed if at least print_frequency_time seconds have passed since last output and the iteration number is a multiple of print_frequency_iter.

(default = 0)

**print_info_string** *(string)*: Enables printing of additional info string at end of iteration output.

This string contains some insider information about the current iteration. For details, look for ”Diagnostic Tags” in the Ipopt documentation.

(default = no)

- no don’t print string
- yes print string at end of each iteration output

**print_level** *(integer)*: Output verbosity level.

Sets the default verbosity level for console output. The larger this value the more detailed is the output.

Range: [0, 12]

(default = 5)
**print_timing_statistics (string):** Switch to print timing statistics.

If selected, the program will print the CPU usage (user time) for selected tasks.

(default = no)

- no: don't print statistics
- yes: print all timing statistics

**probing_cuts (integer):** Frequency k (in terms of nodes) for generating probing cuts in branch-and-cut.

See option 2mir_cuts for the meaning of k.

(default = 0)

**problem_print_level (integer):** Output level for problem manipulation code in Couenne

Range: \([-2, 12]\]

(default = 2)

**pseudocost_mult (string):** Multipliers of pseudocosts for estimating and update estimation of bound

(default = interval_br_rev)

- infeasibility: infeasibility returned by object
- interval_br: width of the interval between bound and branching point
- interval_br_rev: similar to interval_br, reversed
- interval_lp: width of the interval between bound and current LP point
- interval_lp_rev: similar to interval_lp, reversed
- projectdist: distance between current LP point and resulting branches' LP points

**pseudocost_mult_lp (string):** Use distance between LP points to update multipliers of pseudocosts after simulating branching

(default = no)

- no
- yes

**quadrilinear_decomp (string):** Type of decomposition for quadrilinear terms (see work by Cafieri, Lee, Liberti)

(default = rAI)

- bi+tri: Bilinear, THEN trilinear term: \(x_5 = ((x_1 x_2) x_3 x_4)\)
- hier-bi: Hierarchical decomposition: \(x_5 = ((x_1 x_2) (x_3 x_4))\)
- rai: Recursive decomposition in bilinear terms (as in Ryoo and Sahinidis): \(x_5 = ((x_1 x_2) x_3) x_4\)
- tri+bi: Trilinear and bilinear term: \(x_5 = (x_1 (x_2 x_3 x_4))\)

**quality_function_balancing_term (string):** The balancing term included in the quality function for centrality.

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. (Only used if option "mu_oracle" is set to "quality-function".)

(default = none)

- cubic: \(\text{Max}(0, \text{Max}(\text{dual}_{\text{inf}}, \text{primal}_{\text{inf}}) - \text{compl})^3\)
- none: no balancing term is added

**quality_function_centrality (string):** The penalty term for centrality that is included in quality function.

This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the \(x_i\) in the Loqo update rule. (Only used if option "mu_oracle" is set to "quality-function".)

(default = none)
cubed-reciprocal complementarity ∗ the reciprocal of the centrality measure cubed
log complementarity ∗ the log of the centrality measure
none no penalty term is added
reciprocal complementarity ∗ the reciprocal of the centrality measure

**quality_function_max_section_steps** (integer): Maximum number of search steps during direct search procedure determining the optimal centering parameter.

The golden section search is performed for the quality function based mu oracle. (Only used if option “mu_oracle” is set to “quality-function”.)

(default = 8)

**quality_function_norm_type** (string): Norm used for components of the quality function.

(Only used if option “mu_oracle” is set to “quality-function”.)

(default = 2-norm-squared)

1-norm use the 1-norm (abs sum)
2-norm use 2-norm
2-norm-squared use the 2-norm squared (sum of squares)
max-norm use the infinity norm (max)

**quality_function_section_qf_tol** (real): Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).

The golden section search is performed for the quality function based mu oracle. (Only used if option “mu_oracle” is set to “quality-function”.)

Range: [0, 1]

(default = 0)

**quality_function_section_sigma_tol** (real): Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).

The golden section search is performed for the quality function based mu oracle. (Only used if option “mu_oracle” is set to “quality-function”.)

Range: [0, 1]

(default = 0.01)

**random_generator_seed** (integer): Set seed for random number generator (a value of -1 sets seeds to time since Epoch).

(default = 0)

**random_point_perturbation_interval** (real): Amount by which starting point is perturbed when choosing to pick random point by perturbing starting point

(default = 1)

**random_point_type** (string): method to choose a random starting point

(default = Jon)

andreas perturb the starting point of the problem within a prescribed interval
claudia perturb the starting point using the perturbation radius suffix information
jon Choose random point uniformly between the bounds

**read_solution_file** (string): Read a file with the optimal solution to test if algorithms cuts it.

For Debugging purposes only.

(default = no)
yes

recalc_y (string): Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.
This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than
recalc_y_feas_tol. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation
requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is
used by default.
(default = no)
no use the Newton step to update the multipliers
yes use least-square multiplier estimates

recalc_y_feas_tol (real): Feasibility threshold for recomputation of multipliers.
If recalc_y is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.
(default = 1e-06)

redcost_bt (string): Reduced cost bound tightening
This bound reduction technique uses the reduced costs of the LP in order to infer better variable bounds.
(default = yes)
no
yes

reduce_split_cuts (integer): Frequency k (in terms of nodes) for generating reduce_split_cuts cuts in branch-and-cut.
See option 2mir_cuts for the meaning of k.
(default = 0)

red_cost_branching (string): Apply Reduced Cost Branching (instead of the Violation Transfer) – MUST have vt_obj
enabled
(default = no)
no Use Violation Transfer with ∑ ||πia||
yes Use Reduced cost branching with || ∑ πia||

reformulate_print_level (integer): Output level for reformulating problems in Couenne
Range: [-2, 12]
(default = 0)

replace_bounds (string): Indicates if all variable bounds should be replaced by inequality constraints
This option must be set for the inexact algorithm
(default = no)
no leave bounds on variables
yes replace variable bounds by inequality constraints

required_infeasibility_reduction (real): Required reduction of infeasibility before leaving restoration phase.
The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the
infeasibility has been reduced by at least the fraction given by this option.
Range: [0, 1]
(default = 0.9)

residual_improvement_factor (real): Minimal required reduction of residual test ratio in iterative refinement.
If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor,
iterative refinement is aborted.
(default = 1)

**residual_ratio_max** *(real):* Iterative refinement tolerance

Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

(default = 1e-10)

**residual_ratio_singular** *(real):* Threshold for declaring linear system singular after failed iterative refinement.

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

(default = 1e-05)

**resolve_on_small_infeasibility** *(real):* If a locally infeasible problem is infeasible by less than this, resolve it with initial starting point.

(default = 0)

**resto_failure_feasibility_threshold** *(real):* Threshold for primal infeasibility to declare failure of restoration phase.

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is 1e2*tol, where tol is the general termination tolerance.

(default = 0)

**resto_penalty_parameter** *(real):* Penalty parameter in the restoration phase objective function.

This is the parameter rho in equation (31a) in the Ipopt implementation paper.

(default = 1000)

**resto_proximity_weight** *(real):* Weighting factor for the proximity term in restoration phase objective.

This determines how the parameter zeta in equation (29a) in the implementation paper is computed. zeta here is resto_proximity_weight*sqrt(mu), where mu is the current barrier parameter.

(default = 1)

**rho** *(real):* Value in penalty parameter update formula.

Range: [0, 1]

(default = 0.1)

**second_perc_for_cutoff_decr** *(real):* The percentage used when the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically.

(default = -0.05)

**setup_pseudo_frac** *(real):* Proportion of strong branching list that has to be taken from most-integer-infeasible list.

Range: [0, 1]

(default = 0.5)

**sigma_max** *(real):* Maximum value of the centering parameter.

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

(default = 100)

**sigma_min** *(real):* Minimum value of the centering parameter.

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

(default = 1e-06)

**skip_corr_if_neg_curv** *(string):* Skip the corrector step in negative curvature iteration (unsupported!).
The corrector step is not tried if negative curvature has been encountered during the computation of the search
direction in the current iteration. This option is only used if "mu_strategy" is "adaptive".

(default = yes)

<table>
<thead>
<tr>
<th>no</th>
<th>don’t skip</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>skip</td>
</tr>
</tbody>
</table>

**skip_corr_in_monotone_mode** *(string)*: Skip the corrector step during monotone barrier parameter mode (unsupported!).

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive".

(default = yes)

<table>
<thead>
<tr>
<th>no</th>
<th>don’t skip</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>skip</td>
</tr>
</tbody>
</table>

**slack_bound_frac** *(real)*: Desired minimum relative distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the
inequality bounds (together with "slack_bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: \([0, 0.5]\]

(default = 0.01)

**slack_bound_push** *(real)*: Desired minimum absolute distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the
inequality bounds (together with "slack_bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

(default = 0.01)

**slack_move** *(real)*: Correction size for very small slacks.

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack
becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter
determines how large the move should be. Its default value is mach_eps^{3/4}. (See also end of Section 3.5 in
implementation paper - but actual implementation might be somewhat different.)

(default = 1.81899e-12)

**soft_resto_perror_reduction_factor** *(real)*: Required reduction in primal-dual error in the soft restoration phase.

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual
step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least
this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase.

(default = 0.9999)

**solution_limit** *(integer)*: Abort after that much integer feasible solution have been found by algorithm

value 0 deactivates option

(default = maxint)

**solvetrace** *(string)*: Name of file for writing solving progress information.

**solvetracenodefreq** *(integer)*: Frequency in number of nodes for writing solving progress information.

giving 0 disables writing of N-lines to trace file

(default = 100)

**solvetracetimefreq** *(real)*: Frequency in seconds for writing solving progress information.

giving 0.0 disables writing of T-lines to trace file

(default = 5)

**start_with_resto** *(string)*: Tells algorithm to switch to restoration phase in first iteration.
Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure.

(default = no)

- No don’t force start in restoration phase
- Yes force start in restoration phase

**s_max (real):** Scaling threshold for the NLP error.

(See paragraph after Eqn. (6) in the implementation paper.)

(default = 100)

**s_phi (real):** Exponent for linear barrier function model in the switching rule.

(See Eqn. (19) in the implementation paper.)

(default = 2.3)

**s_theta (real):** Exponent for current constraint violation in the switching rule.

(See Eqn. (19) in the implementation paper.)

(default = 1.1)

**tau_min (real):** Lower bound on fraction-to-the-boundary parameter tau.

(This is tau_min in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: [0, 1]

(default = 0.99)

**theta_max_fact (real):** Determines upper bound for constraint violation in the filter.

The algorithmic parameter theta_max is determined as theta_max_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta_max is unacceptable to the filter (see Eqn. (21) in the implementation paper).

(default = 10000)

**theta_min_fact (real):** Determines constraint violation threshold in the switching rule.

The algorithmic parameter theta_min is determined as theta_min_fact times the maximum of 1 and the constraint violation at initial point. The switching rules treats an iteration as an h-type iteration whenever the current constraint violation is larger than theta_min (see paragraph before Eqn. (19) in the implementation paper).

(default = 0.0001)

**time_limit (real):** Set the global maximum computation time (in secs) for the algorithm.

(default = 1000)

**tiny_element (real):** Value for tiny element in OA cut

We will remove "cleanly" (by relaxing cut) an element lower than this.

(default = 1e-08)

**tiny_step_tol (real):** Tolerance for detecting numerically insignificant steps.

If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.

(default = 2.22045e-15)

**tiny_step_y_tol (real):** Tolerance for quitting because of numerically insignificant steps.

If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny_step_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.
(default = 0.01)

tol (real): Desired convergence tolerance (relative).

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled)
NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "constr_viol_tol", and "compl_inf_tol" are met. (This is epsilon_tol in Eqn. (6) in implementation paper). See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to
determine thresholds etc.

(direct = 1e-08)

tree_search_strategy (string): Pick a strategy for traversing the tree

All strategies can be used in conjunction with any of the node comparison functions. Options which affect
dfs-dive are max-backtracks-in-dive and max-dive-depth. The dfs-dive won’t work in a non-convex problem
where objective does not decrease down branches.

(direct = probed-dive)

dfs-dive Dive in the tree if possible doing a depth first search. Backtrack on leaves or when a
prescribed depth is attained or when estimate of best possible integer feasible solution in subtree is
worst than cutoff.

dfs-dive-dynamic Same as dfs-dive but once enough solution are found switch to best-bound and
if too many nodes switch to depth-first.

dive Dive in the tree if possible, otherwise pick top node as sorted by the tree comparison function.

probed-dive Dive in the tree exploring two children before continuing the dive at each level.

top-node Always pick the top node as sorted by the node comparison function

trust_strong (string): Fathom strong branching LPs when their bound is above the cutoff

(direct = yes)

no

yes

trust_strong_branching_for_pseudo_cost (string): Whether or not to trust strong branching results for updating pseudo
costs.

(direct = yes)

no

yes

twoimpl_depth_level (integer): Depth of the B&B tree when to start decreasing the chance of running this algorithm.

This has a similar behavior as log_num_obbt_per_level. A value of -1 means that generation can be done at all
nodes.

(direct = 5)

twoimpl_depth_stop (integer): Depth of the B&B tree where separation is stopped.

A value of -1 means that generation can be done at all nodes

(direct = 20)

two_implied_bt (integer): The frequency (in terms of nodes) at which Couenne two-implied bounds are tightened.

A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne
to generate them at every n nodes of the B&B tree. A negative number -n means that generation should be
attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.

(direct = 0)

two_implied_max_trials (integer): The number of iteration at each call to the cut generator.
use_auxcons \textit{(string)}: Use constraints-defined auxiliaries, i.e. auxiliaries \( w = f(x) \) defined by original constraints \( f(x) - w = 0 \)
  (default = yes)
  
  no
  yes

use_quadratic \textit{(string)}: Use quadratic expressions and related exprQuad class

  If enabled, then quadratic forms are not reformulated and therefore decomposed as a sum of auxiliary variables, each associated with a bilinear term, but rather taken as a whole expression. Envelopes for these expressions are generated through alpha-convexification.
  (default = no)
  
  no Use an auxiliary for each bilinear term
  yes Create only one auxiliary for a quadratic expression

use_semiaux \textit{(string)}: Use semiauxiliaries, i.e. auxiliaries defined as \( w \geq f(x) \) rather than \( w = f(x) \)
  (default = yes)
  
  no Only use auxiliaries assigned with '='
  yes Use auxiliaries defined by \( w \leq f(x), w \geq f(x), \) and \( w = f(x) \)

variable_selection \textit{(string)}: Chooses variable selection strategy
  (default = strong-branching)
  
  lp-strong-branching Perform strong branching with LP approximation
  most-fractional Choose most fractional variable
  nlp-strong-branching Perform strong branching with NLP approximation
  osi-simple Osi method to do simple branching
  osi-strong Osi method to do strong branching
  qp-strong-branching Perform strong branching with QP approximation
  random Method to choose branching variable randomly
  reliability-branching Use reliability branching
  strong-branching Perform strong branching

very_tiny_element \textit{(real)}: Value for very tiny element in OA cut

  Algorithm will take the risk of neglecting an element lower than this.
  (default = 1e-17)

violated_cuts_only \textit{(string)}: Yes if only violated convexification cuts should be added
  (default = yes)
  
  no
  yes

warm_start \textit{(string)}: Select the warm start method

  This will affect the function getWarmStart(), and as a consequence the warm starting in the various algorithms.
  (default = none)
  
  fake_basis builds fake basis, useful for cut management in Cbc (warm start is the same as in none)
  interior_point Warm start with an interior point of direct parent
  none No warm start, just start NLPs from optimal solution of the root relaxation
optimum Warm start with direct parent optimum

`warm_start_bound_frac (real)`: same as `bound_frac` for the regular initializer.
  
  *Range*: [0, 0.5]
  
  *(default = 0.001)*

`warm_start_bound_push (real)`: same as `bound_push` for the regular initializer.
  
  *(default = 0.001)*

`warm_start_init_point (string)`: Warm-start for initial point
  
  Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)
  
  *(default = no)*
  
  - no do not use the warm start initialization
  - yes use the warm start initialization

`warm_start_mult_bound_push (real)`: same as `mult_bound_push` for the regular initializer.
  
  *(default = 0.001)*

`warm_start_mult_init_max (real)`: Maximum initial value for the equality multipliers.
  
  *(default = 1e+06)*

`warm_start_slack_bound_frac (real)`: same as `slack_bound_frac` for the regular initializer.
  
  *Range*: [0, 0.5]
  
  *(default = 0.001)*

`warm_start_slack_bound_push (real)`: same as `slack_bound_push` for the regular initializer.
  
  *(default = 0.001)*

`watchdog_shortened_iter_trigger (integer)`: Number of shortened iterations that trigger the watchdog.
  
  If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure.
  
  *(default = 10)*

`watchdog_trial_iter_max (integer)`: Maximum number of watchdog iterations.
  
  This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.
  
  *(default = 3)*

**Bibliography**


Stochastic Programming (SP) with EMP

Martha Loewe
Michael Ferris
June 2014

Contents

1 Introduction ................................................................. 719
2 The News Vendor .......................................................... 720
  2.1 Uncertain demand: discrete distribution .......................... 720
  2.2 Uncertain demand: continuous distribution ....................... 723
  2.3 Sampling .............................................................. 724
3 Multistage Models ......................................................... 726
4 Chance Constraints ......................................................... 729
  4.1 Single Chance Constraints ............................................. 730
  4.2 Joint Chance Constraints .............................................. 732
  4.3 Individual Chance Constraints ....................................... 734
  4.4 Joint chance constraints vs. individual chance constraints .... 736
  4.5 Penalizing violations of chance constraints ....................... 736
5 Risk Measures ............................................................. 737
  5.1 Expected Value ......................................................... 737
  5.2 Value at Risk (VaR) ..................................................... 739
  5.3 Conditional Value at Risk (CVaR) .................................. 743
6 Summary of keywords and solver configurations .................. 745
7 More on scenarios and output extraction ............................ 747

1 Introduction

This chapter 1 describes the stochastic programming (SP) extension of GAMS EMP. We build a stochastic model based on a deterministic model by defining model parameters to be uncertain. Then GAMS EMP replaces these uncertain parameters by random variables. The distribution of the random variables is controlled by the user. Note that these random variables are not variables in the sense of mathematical optimization, but they can be understood as random parameters.

The chapter is organized as follows: In section The News Vendor the basic principles are introduced with the well known news vendor problem, in section Multistage Models a multi-stage model is discussed, in section Chance Constraints a class of models with chance constraints is presented, while how to model risk measures is the topic of section Risk Measures. In section Summary of keywords and solver configurations a summary of keywords and possible solver configurations is given and finally more details on scenarios and output extraction follow in section More on scenarios and output extraction.

---

1This version created by Martha Loewe and Michael Ferris in June 2014.
2 The News Vendor

Suppose a news vendor wants to maximize his profit $z$. He has to decide how many newspapers to buy from a distributor to satisfy demand $d$. He buys $x$ newspapers at a cost of $c$ per unit. He manages to sell $s$ newspapers at a price of $v$ per paper. If the demand is less than his expectations ($d < x$), then the number $i$ of left-over newspapers are stored in an inventory at a holding cost of $h$ per unit. If the demand exceeds his expectations ($d > x$), then there will be $l$ customers whose demand cannot be met and a penalty of $p$ per unit of unsatisfied demand has to be paid. Assuming that the demand is known the optimization problem can be expressed mathematically as follows:

$$
\max_{x,s,i,l} \quad z = vs - cx - hi - pl \\
\text{s.t.} \quad d = s + l \\
\quad i = x - s \\
\quad x, s, l, i \geq 0.
$$

The objective is to maximize the profit $z$. Here $x, s, i, l$ are the decision variables and the demand $d$ is a known parameter. The demand $d$ equals the sum of the satisfied demand $s$ (papers sold) and the unsatisfied demand $l$ (lost sales). The inventory $i$ equals the difference between the number of papers bought $x$ and those sold $s$. Note that $x, s, l, i \geq 0$.

This can easily be translated into GAMS. We assume that the values for $c, p, h, v$ and $d$ are given. Note that our mathematical formulation is case-sensitive but the GAMS code is not.

```gams
Scalar
   cPurchase costs per unit / 30 /
   pPenalty shortage cost per unit / 5 /
   hHolding cost per unit leftover / 10 /
   vRevenue per unit sold / 60 /
   * Random parameters
   dDemand / 63 /;

Variable
   Z Profit;

Positive Variables
   X Units bought
   I Inventory
   L Lost sales
   S Units sold;

Equations
   Profit, Row1, Row2;

* Profit, to be maximized
   Profit.. Z =e= v*S - c*X - h*I - p*L;
* demand = UnitsSold + LostSales
   Row1.. d =e= S + L;
* Inventory = UnitsBought - UnitsSold
   Row2.. I =e= X - S;

Model nb / all /;
Solve nb max Z use lp;
```

Since the demand $d$ is known there is no uncertainty in this model and the optimal solution is obvious: the best decision is to buy exactly as many newspapers as are demanded. Now we move to more realistic assumptions and consider several examples where the demand is not known from the outset.

2.1 Uncertain demand: discrete distribution

Consider the case when the buying decision should be made before a realization of the demand $d$ becomes known. In our example, the news vendor has to buy newspapers in the morning without knowing the precise demand. However, given his experience he expects the demand to be 45 in 70% of all cases, 40 with a probability of 20% and 50 with a probability of 10%. One way to model such a situation is to regard the demand $D$ as a random variable. By $D$, we denote the demand when viewed as a random variable in order to distinguish it from a particular realization $d$. We assume that the probability distribution of $D$ can be estimated from experience or historical data and is therefore known. Suppose that the set of all realizations of $D$ is finite and given by $\Omega$:

$$
\Omega = \{d_1, d_2, ..., d_{|\Omega|}\}.
$$
Each realization of $D$ is called a scenario. So there is a finite set of scenarios, each associated with probability $p_j$, where $\sum_j p_j = 1$. In our example, the random variable $D$ takes the values $d_1 = 45$, $d_2 = 40$ and $d_3 = 50$ with respective probabilities $p_1 = 0.7$, $p_2 = 0.2$ and $p_3 = 0.1$. The decision $x$ has to be made before the realization of the demand $D$ is known.

After the news vendor has made the decision of how many newspapers $x$ to buy on a particular day the actual demand is disclosed. He may have bought more that he can sell and may have to store the surplus in his inventory or he may not have bought enough and some demand may remain unsatisfied. We can translate this situation into a model with two stages: in stage 1 a decision $x$ is made without knowing the future, then one realization of the future unfolds and in stage 2 a second period decision $s, i, l$ is made that attempts to react to the new situation. The action taken in stage 2 is called recourse. The key idea in this approach is the evolution of information.

We express this two stage model mathematically in the following way:

$$\begin{align*}
\text{Max}_x \quad & Z(x, D) = -cx + \mathbb{E}[Q(x, D)], & x \geq 0, \quad (2)
\end{align*}$$

where

$$\begin{align*}
Q(x, D) &= \text{Max}_{s,i,l} \quad vs_D - hi_D - pl_D \\
&\text{s.t.} \quad x - s_D - i_D \quad = 0 \\
&\quad s_D + l_D = D \\
&\quad s_D, i_D, l_D \geq 0. \quad (3)
\end{align*}$$

Given the uncertainty of the demand we aim to maximize the expected value of the profit, denoted by $\mathbb{E}[Z(x, D)]$. The expected value of the profit is the profit on average. Note that since we have a finite number of scenarios and their probabilities are known, the expected value of the profit $\mathbb{E}[Z(x, D)]$ can be expressed as a weighted sum:

$$\begin{align*}
\mathbb{E}[Z(x, D)] &= -cx + \sum_{k=1}^3 p_k Q(x, d_k). \quad (4)
\end{align*}$$

Notice that we have separated the original optimization problem into two different problems that are solved at two stages. The decision variable in the first problem (at stage 1) is $x$ and it is the vector $y_D = (s_D, i_D, l_D)$ in the second problem (at stage 2). We introduce additional notation to explicitly represent the underlying structure of the problem. Let

$$\begin{align*}
q_D &= \begin{bmatrix} v \\ -h \\ -p \end{bmatrix}, & T_D = \begin{bmatrix} I \\ 0 \end{bmatrix}, & W_D = \begin{bmatrix} -I & -I & 0 \\ I & 0 & I \end{bmatrix}, & h_D = \begin{bmatrix} 0 \\ D \end{bmatrix}.
\end{align*}$$

Using this new notation the second stage optimization problem from (3) becomes:

$$\begin{align*}
Q(x, D) &= \text{Max}_{y_D} \quad q_D^T y_D \\
&\text{s.t.} \quad T_D x + W_D y_D = h_D, \quad y_D \geq 0. \quad (5)
\end{align*}$$

Note that in our specific example $T_D \equiv T$, $W_D \equiv W$ and $q_D \equiv q$ since they are independent of the demand $D$. The general two stage optimization problem is given here:

$$\begin{align*}
\text{Max}_{x \in \mathbb{R}^n} \quad & Z(x, \zeta) = -c^T x + \mathbb{E}[Q(x, \zeta)] \\
&\text{s.t.} \quad Ax = b, \quad x \geq 0, \quad (6)
\end{align*}$$

where

$$\begin{align*}
Q(x, \zeta) &= \text{Max}_{y \in \mathbb{R}^m} \quad q^T \zeta y \\
&\text{s.t.} \quad T_\zeta x + W_\zeta y = h_\zeta, \quad y \geq 0, \quad (7)
\end{align*}$$
\[ \zeta = (q, h, T, W), \]

the data of the second stage problem. Note that in our news vendor example there are no first stage equations \( Ax = b \). Table 1 summarizes the two stages in our example.

### Table 1: Two stages in news vendor problem

<table>
<thead>
<tr>
<th>1st stage decision variable</th>
<th>2nd stage decision variable ( \Omega = {d_1, d_2, d_3} )</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_{d_1} = (s_{d_1}, l_{d_1}, i_{d_1}) )</td>
<td>( y_{d_2} = (s_{d_2}, l_{d_2}, i_{d_2}) )</td>
<td>Scenario 1: ( p_1 = 0.7 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( y_{d_3} = (s_{d_3}, l_{d_3}, i_{d_3}) )</td>
<td>Scenario 2: ( p_2 = 0.2 )</td>
</tr>
<tr>
<td>( y_{d_3} = (s_{d_3}, l_{d_3}, i_{d_3}) )</td>
<td>( y_{d_3} = (s_{d_3}, l_{d_3}, i_{d_3}) )</td>
<td>Scenario 3: ( p_3 = 0.1 )</td>
</tr>
</tbody>
</table>

The problem as stated in (6), (7) can be converted to a single large scale optimization problem. The extended form is given below:

\[
\begin{align*}
\text{Max}_{x} & \quad -c^T x + \sum_i p_i [\text{Max}_y q^T \zeta_i y] \\
\text{s.t.} & \quad Ax = b \\
& \quad T_x x + W \zeta x = h \\
& \quad x, y \geq 0.
\end{align*}
\]

Finally, we present the problem in matrix form, a notation that has the advantage of explicitly displaying the structure of the second stage:

\[
\begin{align*}
\text{Max}_{x,y} \quad & -c^T x + \sum_i p_i q^T \zeta_i y \\
\text{s.t.} \quad & \begin{pmatrix} A & T_{\zeta_1} & W_{\zeta_1} \\ T_{\zeta_2} & W_{\zeta_2} \\ \vdots & \vdots \\ T_{\zeta_m} & W_{\zeta_m} \end{pmatrix} \begin{pmatrix} x \\ y_{\zeta_1} \\ y_{\zeta_2} \\ \vdots \\ y_{\zeta_m} \end{pmatrix} = \begin{pmatrix} b \\ h_{\zeta_1} \\ h_{\zeta_2} \\ \vdots \\ h_{\zeta_m} \end{pmatrix}, \quad x, y_{\zeta} \geq 0.
\end{align*}
\]

To formulate this 2-stage-model in GAMS we introduce the notions of a core problem and annotations. The core problem defines \( q, h, T, W \) as parameters instead of having them as random variables. It can be written as follows:

\[
\begin{align*}
\text{Max}_{x,y} \quad & -c^T x + q^T y \\
\text{s.t.} \quad & \begin{pmatrix} A & 0 \\ W \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ h \end{pmatrix}, \quad x, y \geq 0.
\end{align*}
\]

The annotations give details about the various realizations of the random variable \( \zeta \), the scenarios.

For our news vendor example the core model is precisely the model given in section The News Vendor. Then we add the annotations: the distribution of the random variable (in our case the demand which is part of the variable \( h \)) has to be specified as random and every variable and equation must be assigned to one of the two stages. This information is introduced by writing a text file named %emp.info%. Here is the file for our model:

```plaintext
1 file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
2 $onput
3 randvar d discrete 0.7 45 0.2 40 0.1 50
4 stage 2 I L S d
5 stage 2 Row1 Row2
6 $offput
7 putclose emp;
```

First, we define the parameter \( d \) to be a random variable (randvar) with a discrete distribution: with probability 0.7 it takes a value of 45, with probability 0.2 it takes a value of 40, and with probability 0.1 it takes a value of 50. Then the variables and equations of stage 2 are listed. The variables and equations not listed in the emp.info% file are automatically
assigned to a stage, with a default assumed to be stage 1. Note that the objective variable (in our case \( Z \)) and the profit equation are thus assigned to the highest stage specified (stage 2 in this example). Observe that \( Z \) is in fact a random variable since it is a function of the random variable \( D \). As such it cannot be maximized, EMP implicitly maximizes the \textit{expected value} of \( Z \). We believe this might lead to some confusion since the expected value \( E[Z] \) belongs to stage 1. We show later how to specify more clearly the fact that we are maximizing \( E[Z] \). All keywords that can be used in the emp.info file are introduced in subsequent examples and summarized in section \textit{Summary of keywords and solver configurations}.

Given the probability distribution the solvers of stochastic programming models create various scenarios and evaluate them. Where should the details of these scenarios be stored (or communicated to the modeler)? In deterministic models the modeler does not need to specify how the output should be stored. Using EMP for stochastic programming the modeler can store the results for each scenario in standard parameters. Here is how it is done:

1. Set \texttt{scen} Scenarios / s1*s3 /;
2. Parameter
3. \texttt{s_d(scen)} Demand realization by scenario
4. \texttt{s_x(scen)} Units bought by scenario
5. \texttt{s_s(scen)} Units sold by scenario;
6. Set \texttt{dict} / scen .scenario.''
7. \texttt{d .randvar .s_d}
8. \texttt{s .level .s_s}
9. \texttt{x .level .s_x} /;
10. solve nb max z use emp scenario dict;

The size of the set \texttt{scen} defines the maximal number of scenarios we are willing to store results for. The three dimensional set \texttt{dict} contains mapping information between symbols in the model (in the first position) and symbols to store solution information (in the third position), and the type of storing (in the second position). An exception to this rule is the tuple with label \texttt{scenario} in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. In our example, we want to store the realization for each scenario for the random variable \( d \) in the parameter \texttt{s_d} and the levels of the variables \( s \) and \( x \) in the parameters \texttt{s_s} and \texttt{s_x} respectively. A more detailed description on scenarios and how this set works can be found in section \textit{More on scenarios and output extraction}.

Finally, the solve statement needs to be adjusted: we use emp instead of lp and add \texttt{scenario dict} to indicate that a stochastic problem should be solved.

### 2.2 Uncertain demand: continuous distribution

Now let’s assume that the random variable \( D \) has a continuous distribution, say a Normal distribution with mean 45 and standard deviation 10. This problem has the same structure as the problem discussed in section \textit{Uncertain demand: discrete distribution}, where the random variable \( D \) has a discrete distribution. The difference is that the set \( \Omega \), that is the set of all realizations of \( D \), contains an infinite number of scenarios. There are various ways of modeling this, one of which is a sampling procedure implemented in the solver. A finite number of scenarios is generated to approximate \( \Omega \) and thus the problem is converted to a problem as discussed in section \textit{Uncertain demand: discrete distribution}.

In GAMS we can model a continuous distribution of the random variable by changing the \texttt{randvar} line in the emp.info file in the following way:

```
randvar d normal 45 10
```

Note that currently only the solver LINDO has implemented a sampling procedure for parametric distributions. More details about sampling are given in the next section. In Table 2 all parametric distributions that can be modeled are listed.

### Table 2: Parametric distributions supported by LINDO

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Parameter 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta</td>
<td>shape 1</td>
<td>shape 2</td>
<td></td>
</tr>
<tr>
<td>Cauchy</td>
<td>location</td>
<td>scale</td>
<td></td>
</tr>
<tr>
<td>Chi-Square</td>
<td>deg. of freedom</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We have to inform GAMS that we want the solver LINDO to be used. There are two ways to do this: we either state in the command line `emp=lindo` or we insert the following statement before the `solve` statement in the GAMS file:

```gams
option emp = lindo;
```

### 2.3 Sampling

Currently only the solver LINDO has the ability to perform sampling for continuous distributions. It generates 6 samples by default. There are three ways to customize sampling: 1) adding additional information to the `emp.info` file, 2) generating a sample with the LINDO library `lsadclib` and then using this sample in any solver with EMP capabilities and 3) setting various options in the solver LINDO. We will discuss each method in more detail in the remainder of this section.

#### Customizing sampling with EMP

Observe that customizing sampling with EMP is currently experimental, feedback is requested, and possible changes might occur when its notation becomes fixed. Currently, EMP provides two keywords to enable users to customize sampling: `sample` and `setSeed`. The keyword `sample` allows the user to customize the size of the sample in the `emp.info` file. Consider the following example:

```
gams
randvar d normal 45 10
sample d 9
```

The second line determines the size of the sample of the distribution of the random variable \( D \) to be 9. Note that currently the LINDO Sampling library is used for this sampling. Users who don’t have a valid LINDO license are limited to the Normal and Binomial distributions with a maximum sample size of 10.

The keyword `sample` also offers the possibility to determine a mathematical variance reduction method to be applied. Variance reduction is a procedure used to increase the precision of the estimated values from the distribution. LINDO provides three methods for reducing the variance: Monte Carlo sampling, Latin Square sampling and Antithetic sampling.
Consider a stochastic model with four random variables: \( E, F, G \) and \( H \). Assume that \( E \) follows a Normal distribution with mean 23 and standard deviation 5, \( F \) follows a Normal distribution with mean 37 and standard deviation 8, \( G \) is uniformly distributed on the interval \([0,1]\) and \( H \) is binomially distributed with \( n = 100 \) and \( p = 0.55 \). We want for each random variable a sample size of 10 and we want to apply the three variance reduction methods in the following way: LINDO shall use Antithetic sampling for \( E \) and \( F \), Monte Carlo sampling for \( G \) and Latin Square sampling for \( H \). To model this we insert the following lines in the emp.info file:

1. randvar e normal 23 5
2. randvar f normal 37 8
3. randvar g uniform 0 1
4. randvar h binomial 100 55
5. sample e f 10 method1
6. sample g 12 method2
7. sample h 8 method3

In lines 1 to 4 the random variables and their distributions are defined, in lines 5 to 7 details about the sampling procedures are given. Note that the keyword sample can take more than one random variable (line 5) if the sample size and the variance reduction method for these random variables are the same. We need to add the following lines before the solve statement to specify the content of method1, method2 and method3 (we assume that the name of the model is nb):

1. $onecho > lindo.opt
2. 2 SVR_LS_ANTITHETIC=method1
3. 3 SVR_LS_MONTECARLO=method2
4. 4 SVR_LS_LATINSQUARE=method3
5. $offecho
6. nb.optfile=1;

If the Latin Square sampling should also be used for \( E \) and \( F \), we would simply change the emp.info file to replace the label method 1 with the label method 3. For more details on variance reduction methods please consult the LINDO manual.

A further way to customize the sampling procedure is the keyword setSeed:

setSeed <seed>

This sets the seed for the random number generator of the sampling routines called using the sample keyword. If setSeed is used in the emp.info file, the seed is set once before we generate all samples. Please note that setSeed only works with a valid LINDO license.

### Separating sampling and solving

A user may want to sample from a distribution with the LINDO system and solve the model with another solver, say DE. This is possible with the sampling routines from the LINDO library lsadclib. We could solve the news vendor model by first drawing a sample from a Normal distribution with mean 45 and standard deviation 10 and then using the sample in the emp.info file. Here is the GAMS code for sampling from a Normal distribution where the sample size is 9:

1. $funclibin msllib lsadclib
2. function setSeed / msllib.setSeed /
3. sampleNormal / msllib.sampleLSnormal /
4. getSampleValues / msllib.getSampleValues /;
5. scalar k;
6. k = sampleNormal(45,10,9);
7. set g /1*9/;
8. parameter svi(g);
9. loop(g,
10. svi(g) = getSampleValues(k);
11. );
12. display svi;

The directive in the first line makes the LINDO sampling library available, msllib is the internal library name. For further details and a list of the available distributions please consult the LINDO manual.

In the following lines we demonstrate how the sample is used in the emp.info file:
1 file emp / 'emp.info' /; put emp '* problem %gams.i%'/;
2 put 'randvar d discrete '; loop(g, put (1/card(g)) ' ' sv1(g) ' ');
3 $onput
4 stage 2 I L S d
5 stage 2 Row1 Row2
6 $offput
7 putclose emp;

Line 2 states that the random variable $D$ follows a discrete distribution and the probabilities and values are taken from the previously generated sample. The other lines of the emp.info file remained unchanged.

**Sampling options in the LINDO solver**

There are some customizable sampling options in LINDO. The user could control the number of sampled scenarios by setting any of the following LINDO/SP options in the lindo.opt file:

- **STOC_NSAMPLE_PER_STAGE** - list of sample sizes per stage (starting at stage 2)
- **STOC_NSAMPLE_SPAR** - common sample size per stochastic parameter
- **STOC_NSAMPLE_STAGE** - common sample size per stage

For example, we could insert the following three lines before the `solve` statement:

```gams
option emp = lindo;
$echo STOC_NSAMPLE_STAGE = 100 > lindo.opt
nb.optfile = 1;
```

The first line tells GAMS to solve the `emp` modeltype using the LINDO solver, the second line writes "STOC_NSAMPLE_STAGE = 100" to the lindo.opt file, which indicates the solver to generate 100 samples per stage, and the third line informs GAMS to use the solver option file (i.e. lindo.opt). For more details about LINDO options please consult the user manual.

### 3 Multistage Models

In models with more than two stages the same principle applies as in 2-stage models: new information is revealed at the beginning of the stage and recourse decisions or adjustments are made after this information is available. At the point where decisions are made only outcomes of the current stage and previous stages are available. In (11) this logic is pictured schematically.

![Diagram](https://via.placeholder.com/250)

Observe that random variables which are realized in stage $k$ are fixed parameters in stage $k + 1$; stage 1 random variables are in fact simply given deterministic values.

Consider an inventory problem where the decision must be made at the end of each week how many hats should be bought in order to satisfy the stochastic demand in the following week with the aim to maximize the profit. We assume that the stochastic demand can be modeled using a Gamma distribution. The planning horizon is 3 weeks. Before the first week starts an initial purchase decision has to be made and the goods are stored in the inventory for use in week 1. At this point only the distribution of the demand of the first week is known. During the first week the actual demand is revealed and some items that were stored in the inventory are sold. Some items may be left over; they are stored as the inventory for the second week. In addition, a purchase decision for the second week has to be made given the size of the inventory and the distribution of the demand in the second week. Again, the actual demand is revealed in the course of the second week. The same holds for the third week.
We will model the problem with 4 stages, where the first stage corresponds to the preparation time before the first week, the second stage corresponds to decisions made in the first week, the third stage corresponds to decisions made the second week and the fourth stage corresponds to decisions made in the third week. Let \( t \) denote the stages. Note that while the stages range from \( t = 1 \) to \( t = 4 \), demand variables are realized only at \( t = 2 \) to \( t = 4 \). Let \( y_t \) be the amount bought and \( i_t \) the amount stored at the end of each stage and let \( D_t \) denote the demand in each week and \( s_t \) the amount sold each week. Note that we denote the demand with a capital \( D \) since it is a random variable. Let \( \alpha = 10 \) be the cost per hat bought, \( \beta = 20 \) the revenue per hat sold and \( \delta = 4 \) the storage cost per unit. Further, in the storage facility a maximum of \( \kappa = 5000 \) hats can be stored.

As discussed in section Uncertain demand: continuous distribution above the solvers use a sampling procedure to approximate a problem with a continuous random variable such as a Gamma distributed random variable by a problem with a discrete distribution. Figure 1 illustrates the stages assuming a sample size of 6.

The stochastic optimization problem can be expressed as follows:

\[
\begin{align*}
\text{Max}_{s_t, y_t, i_t} \quad z &= -\alpha y_1 - \gamma i_1 + E[\max \beta s_2(D_2) - \alpha y_2(D_2) - \delta i_2(D_2) + \cdots + E[\max \beta s_4(D_4) - \alpha y_4(D_4) - \delta i_4(D_4)]] \\
\text{s.t.} \quad &i_1 = y_1 \\
&i_{t-1} + y_t = s_t + i_t \\
&s_t \leq i_{t-1} \\
&s_t \leq D_t \\
&i_t \leq \kappa \\
&y_1, i_1, s_1, y_4, i_4 \geq 0,
\end{align*}
\]

where \( t = 2, \ldots, 4 \) and \( D_t \) follows a Gamma distribution. Note that \( s_t, y_t \) and \( i_t \) depend on the realization of \( D_t \).

The GAMS code follows.

```gams
1 set t /1*4/;
2 set st(t) *stages where sales occur* /2*4/ ;
3 4 scalars
5 kappa capacity of storage building
```
alpha cost per unit bought
beta revenue per unit sold
delta cost per unit stored at the end of time period;

parameters
k "shape of demand (1st parameter of gamma distribution)"
d_theta(st) "scale of demand (2nd parameter of gamma distribution)"
d(st) "demand";

positive variables
y(t) units to be bought in time period t
i(t) ending inventory in period t
s(t) units sold in time period t;

free variable profit;

Now assign all the data
k = 16;
d(st) = k * d_theta(st);
display d;
kappa = 5000;
alpha = 10;
beta = 20;
delta = 4;

equations
profit eq profit to be max
Bal eq(t) balance equation
Sales eq1(st) sales cannot exceed demand
Sales eq2(t) sales cannot exceed inventory of previous time period;

profit eq.. profit =e= sum(t, beta * s(t)$st(t) - alpha * y(t) - delta * i(t));

Bal eq(t).. i(t-1) + y(t) =e= s(t)$st(t) + i(t);
Sales eq1(st).. s(st) =l= d(st);
Sales eq2(t)$st(t).. s(t) =l= i(t-1);
i.up(t) = kappa;
model inventory /all/;

file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
put emp; emp.nd=6;
put "randvar d('2') gamma ", k d_theta('2') /;
put "randvar d('3') gamma ", k d_theta('3') /;
put "randvar d('4') gamma ", k d_theta('4') /;
$onput
stage 1 y('1') i('1') Bal eq('1')
stage 2 y('2') d('2') i('2') Bal eq('2') Sales eq1('2') Sales eq2('2')
stage 3 y('3') d('3') i('3') Bal eq('3') Sales eq1('3') Sales eq2('3')
stage 4 y('4') d('4') i('4') Bal eq('4') Sales eq1('4') Sales eq2('4')
$offput
putclose emp;

Set scen Scenarios / s1*s1000 /;
Parameter
s_d(scen,st) Demand realization by scenario
s_y(scen,t) Units bought by scenario
s_i(scen,t) Units sold by scenario
s_s(scen,t) Units stored by scenario;

Set dict /
scen .scenario.''
d .randvar .s_d
y .level .s_y
i .level .s_i /

solve inventory max profit using emp scenario dict;
display s_d, s_s, s_y, s_i;

Observe that in the core problem the values of the demand d(t) are replaced by the expected values of the random variable D_t which follows a Gamma distribution. Note that as expected, in stage 1 we have only the variables y and i, but no variables s and d. Note that currently only the solver LINDO can solve models with parametric distributions (see sections Uncertain demand: continuous distribution and Sampling). As discussed in section Sampling above the user could use the LINDO system to generate samples and then solve the optimization problem with another solver. Here is the GAMS code that allows the user to draw samples for the inventory
model (the sample size is 10). In this code we generate samples from 3 different distributions \( f, g \) and \( h \) and store the samples in the parameters \( sv1 \), \( sv2 \) and \( sv3 \).

```plaintext
1 $funclibin msllib lsadclib
2 function setSeed / msllib.setSeed /
3 sampleGamma / msllib.sampleLSGamma /
4 getSampleValues / msllib.getSampleValues /;
5
6 $if not set ss $set ss 10
7 scalar f, g, h;
8 f = sampleGamma(16,208.3125,%ss%);
9 g = sampleGamma(16,312.5,%ss%);
10 h = sampleGamma(16,125,%ss%);
11
12 set j /1*%ss%/;
13 parameter sv1(j),sv2(j), sv3(j);
14
15 loop(j,
16 sv1(j) = getSampleValues(f); );
17 display sv1;
18
19 loop(j,
20 sv2(j) = getSampleValues(g); );
21 display sv2;
22
23 loop(j,
24 sv3(j) = getSampleValues(h); );
25 display sv3;
```

The user has to modify the `emp.info` file to use the generated samples \( sv1 \), \( sv2 \) and \( sv3 \) with probabilities \( \frac{1}{J} \) to solve the model:

```plaintext
1 file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
2 put emp; emp.nd=6;
3 put "randvar d('2') discrete "; loop(j, put (1/card(j)) ' ' sv1(j) ' ');
4 put "randvar d('3') discrete "; loop(j, put (1/card(j)) ' ' sv2(j) ' ');
5 put "randvar d('4') discrete "; loop(j, put (1/card(j)) ' ' sv3(j) ' ');
6 $onput
7 stage 1 y('1') i('1') Inv1 eq
8 stage 2 y('2') d('2') s('2') i('2') Bal_eq('2') Sales_eq1('2') Sales_eq2('2')
9 stage 3 y('3') d('3') s('3') i('3') Bal_eq('3') Sales_eq1('3') Sales_eq2('3')
10 stage 4 y('4') d('4') s('4') i('4') Bal_eq('4') Sales_eq1('4') Sales_eq2('4')
11 $offput
12 putclose emp;
```

Note that lines 3-5 are newly inserted in order to use the generated samples; the stages remain unchanged.

In case the modeler wants to use large samples with limited computing capability, LINDO offers the option to approximate the solution with a Benders decomposition algorithm. The following lines can be inserted to achieve this:

```plaintext
1 $onecho > lindo.opt
2 STOC_MAX_NUMSCENS = 1000000
3 STOC_NSAMPLE_STAGE = 40
4 STOC_METHOD = 1
5 $offecho
6 inventory.optcr=1e-4;
```

In line 2 we ensure that the maximum number of scenarios is large enough, the option in line 3 states the sample size and the option in line 4 determines the stochastic method to be used (1 means Nested Benders decomposition). For further details on LINDO options please consult the LINDO user manual. We add line 6 to ensure that the gap between the true solution and the approximation is reasonably small.

### 4 Chance Constraints

A major class of problems in stochastic modeling involves chance constraints. The goal in these problems is to make an optimal decision prior to the realization of random data while allowing the constraints to be violated with a certain probability.
Let $\tilde{A} \in \mathbb{R}^{m \times n}$ be a random matrix and let $\tilde{b} \in \mathbb{R}^m$ be a random vector. Then a general stochastic linear program can be written as:

$$\begin{align*}
\text{Min}_x & \quad c^T x \\
\text{s.t.} & \quad \tilde{A}x \leq \tilde{b} \\
& \quad x \geq 0.
\end{align*}$$

(13)

All feasible solutions satisfy all constraints simultaneously. The distinctive feature of a program with chance constraints is that we require $\tilde{A}x \leq \tilde{b}$ to be satisfied not in all cases but only with some prescribed probability $p$, where $0 < p \leq 1$. Note that often we are interested in the risk tolerance $\varepsilon$, $\varepsilon$ being the probability that a constraint is not satisfied. So $p = 1 - \varepsilon$.

Throughout this chapter we use $p$ and $1 - \varepsilon$ interchangeably. A general stochastic linear problem with chance constraints can be written as follows:

$$\begin{align*}
\text{Min}_x & \quad c^T x \\
\text{s.t.} & \quad P(\tilde{A}x \leq \tilde{b}) \geq p \\
& \quad x \geq 0.
\end{align*}$$

(14)

Solvers convert a problem like (14) into a mixed-integer problem (MIP) first and then solve the MIP equivalent. They introduce a vector with binary variables, say $y_k \in \mathbb{R}^m$, for each scenario $k \in S$. The binary variables take value 1 if the corresponding constraint is satisfied in a scenario and 0 otherwise. A scenario-based formulation of the chance-constrained stochastic linear program (14) can be written as:

$$\begin{align*}
\text{Min}_x & \quad c^T x \\
\text{s.t.} & \quad A^kx \leq b^k + M^k(1 - y_k) \\
& \quad \sum_{k \in S} y_k \geq p \times |S| \\
& \quad x \geq 0, \quad y \in (0, 1)^{|S|},
\end{align*}$$

(15)

where $M^k \in \mathbb{R}^m$ is a chosen big-M vector. The entries of the vector $M^k$ should be chosen such that it does not cut off any feasible solution if an entry of $y_k = 0$.

We will first discuss the special case where the random matrix $\tilde{A}$ is a one-row vector $\tilde{a} \in \mathbb{R}^n$, the random vector $\tilde{b}$ is a single random variable and we have only one chance constraint. Then we will move on to the two ways to model problems with multiple chance constraints: using joint chance constraints and using individual chance constraints. Joint chance constraints require all constraints to be satisfied simultaneously with a given probability. Individual chance constraints require each constraint to be satisfied with a given probability independent of other constraints. We discuss joint chance constraints in section Joint Chance Constraints, individual chance constraints in section Individual Chance Constraints and compare them in section Joint chance constraints vs. individual chance constraints. Finally, in section Penalizing violations of chance constraints the option to penalize violation of constraints is introduced.

### 4.1 Single Chance Constraints

Consider the following chance-constrained stochastic linear problem:

$$\begin{align*}
\text{Min}_x & \quad c^T x \\
\text{s.t.} & \quad P(\tilde{a}x \leq \tilde{b}) \geq p \\
& \quad x \geq 0,
\end{align*}$$

(16)

where $\tilde{a}$ is a random row vector and $\tilde{b}$ is a random variable. Given a set of scenarios $S$ let each scenario $k \in S$ be realized with probability $\pi^k$. The corresponding realizations of $\tilde{a}$ and $\tilde{b}$ are denoted by $a^k$ and $b^k$ respectively. The inequalities of the $|S|$ scenarios may be expressed as follows:

$$a^1x \leq b^1$$
$$a^2x \leq b^2$$
$$\vdots$$
$$a^{|S|}x \leq b^{|S|}$$

(17)
If each scenario is equally likely to be realized then the decision variable $x$ must be chosen such that the inequality is satisfied in at least $p \times |S|$ scenarios.

Here is an example with two decision variables and 4 scenarios where each scenario is equally likely to be realized (i.e. $\pi^k = \frac{1}{4}$):

$$\text{Min } x_1 + x_2$$
$$\text{s.t. } P(\omega x_1 + x_2 \geq 7) \geq 0.75, \quad \omega \in \Omega = \{1, 2, 3, 4\}$$

Note that in this example $b$ is fixed at 7. Note further that there are four scenarios since there are four possible realizations of $\omega$. Since $p = 0.75$ and each scenario is equally likely to be realized we need to choose $x_1$ and $x_2$ such that the inequality is satisfied in at least 3 scenarios. The inequalities for the four scenarios are given below:

$$k = 1: \quad \omega^1 x_1 + x_2 \geq 7$$
$$k = 2: \quad \omega^2 x_1 + x_2 \geq 7$$
$$k = 3: \quad \omega^3 x_1 + x_2 \geq 7$$
$$k = 4: \quad \omega^4 x_1 + x_2 \geq 7.$$  

The MIP equivalent is given below:

$$\text{Min } x_1 + x_2$$
$$\text{s.t. } 1 \quad x_1, x_2 \geq 0$$
$$2 \quad 1 x_1 + x_2 \geq 7 - M(1-y_1)$$
$$3 \quad 2 x_1 + x_2 \geq 7 - M(1-y_2)$$
$$4 \quad 3 x_1 + x_2 \geq 7 - M(1-y_3)$$
$$5 \quad 4 x_1 + x_2 \geq 7 - M(1-y_4)$$
$$6 \quad cc_1 = 1 - \sum_{k=1}^{4} \pi^k y_k, \quad k = 1, \ldots, 4, \pi^k = \frac{1}{4}$$

Observe that the first four constraints cover the four possible scenarios with $\omega$ taking the values 1, 2, 3 and 4 respectively. On the right handside we introduce a big-M factor and $y_k$, a binary indicator variable. $y_k$ takes the value 1 if the constraint is satisfied and 0 otherwise. A new variable, $cc_1$, is introduced in the fifth constraint representing the probability that the constraint is violated. If $cc_1 = 0$ the sum equals 1, indicating that the constraint is satisfied in all four scenarios. If $cc_1 = 0.25$ the constraint remains unsatisfied in one scenario out of four (for this scenario $y_k = 0$).

The problem can be modeled in GAMS as follows:

1 Scalar
2 on / 1 /;
3 4 Variable Z Objective;
5 Positive Variables X1, X2;
6 7 Equations OBJ, E1;
8 9 OBJ.. Z =e= X1 + X2;
10 E1.. on*X1 + X2 =g= 7;
11 12 Model sc / all /;
13 14 file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
15 $onput
16 randvar on discrete 0.25 1 0.25 2 0.25 3 0.25 4
17 chance E1 0.75
18 $offput
19 putclose emp;
20 21 Set scen scenarios / s1*s12 /;
22 Parameter
23 s_on(scen)
24 s_l (scen)
As introduced before we start with a core model and then add annotations and output handling information. Currently, there are no stages unlike in the problems with recourse. Observe that we introduced the new keyword \textit{chance}. The line \texttt{chance} $E_1 \ 0.75$ specifies that the constraint $E_1$ must hold for at least 75\% of all scenarios. We can verify that this requirement has been enforced by checking in the output file the level value of the constraint $e_{1,1}$, and seeing the first scenario constraint is violated at the solution of the chance constrained problem.

Note that the default value of $M$ in the solvers Lindo and DE is 10000. Currently it can only be customized in DE. We could insert the following five lines before the solve statement to set the value of $M$ to 1000:

\begin{verbatim}
option emp = de;
$onecho > de.opt
ccreform bigM 1e3
$offecho
sc.optfile = 1;
\end{verbatim}

Alternatively, the first line could be replaced by placing \texttt{emp=de} on the command line. Note that it is important that \texttt{optca} and \texttt{optcr} are assigned the appropriate values. The GAMS default for the absolute gap \texttt{optca} is 0 and the default for the relative gap \texttt{optcr} is 0.1.

Observe that in addition to converting the chance-constraint problem to a MIP using $M$ the solver DE offers two further options to solve chance-constraint problems: a reformulation using a convex hull and a reformulation using indicator variables and indicator constraints. The following line indicates that a convex hull with $M = 1000$ and $\varepsilon = 0.00001$ is to be used:

\texttt{ccreform cHull 1e3 1e-6}

For indicator variables and constraints we use the following line:

\texttt{ccreform indic}

Note that currently only the solver CPLEX supports indicator variables, so the resulting reformulated problem has to be solved with CPLEX.

### 4.2 Joint Chance Constraints

In this section we discuss the general stochastic linear problem with chance constraints as introduced in (14) assuming that $p$ is the preassigned probability that all constraints are simultaneously satisfied.

\begin{equation}
\begin{align*}
\text{Min} & \quad c^T x \\
\text{s.t.} & \quad P(\bar{A}x \geq \bar{b}) \geq p \\
\end{align*}
\end{equation}

Consider the example (18) from the previous section extended by one constraint, so that we have two decision variables and two constraints. The random data follow discrete uniform distributions so each scenario is equally likely.

\begin{equation}
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad P(\omega_1 x_1 + x_2 \geq 7; \omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \quad (\omega_1, \omega_2) \in \Omega
\end{align*}
\end{equation}
where
\[ \Omega = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3), (4,1), (4,2), (4,3)\} \] (23)
and
\[ \pi^k = \pi(\omega_1, \omega_2) = \frac{1}{12} \quad \text{for all} \quad (\omega_1, \omega_2) \in \Omega. \] (24)

Note that in this example \( b \) is not random and we have 12 scenarios that are all equally likely. The MIP equivalent is given below:

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\
& \quad 2x_1 + x_2 \geq 7 - M(1 - y_2) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_2) \\
& \quad \vdots \\
& \quad 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\
& \quad 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\
& \quad cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12, \pi^k = \frac{1}{12} \\
& \quad x_1, x_2 \geq 0 \\
& \quad 0 \leq cc_1 \leq (1 - 0.6) \\
& \quad y_k \in (0, 1). 
\end{align*}
\] (25)

Note that the first set of constraints cover the 12 scenarios, where each scenario has two constraints. The other constraints are similar to those introduced in (20).

The corresponding GAMS model follows.

1 Scalar
2 om1 / 1 / 
3 om2 / 1 /;
4 Variable 2 Objective; 
5 Positive Variables X1,X2; 
6 Equations OBJ, E1, E2; 
7 OBJ.. Z =e= X1 + X2; 
8 E1.. om1*X1 + X2 =g= 7; 
9 E2.. om2*X1 + 3*X2 =g= 12; 
10 Model sc / all /;
11
12 file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
13 $onput
14 randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4 
15 randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3 
16 chance E1 E2 0.6 
17 $offput
18 putclose emp;
19
20 Set scen scenarios / s1*s12 /;
21 Parameter 
22 s_om1(scen) 
23 s_om2(scen) 
24 x1_l(scen) 
25 x2_l(scen) 
26 x1_m(scen) 
27 e1_l(scen) 
28 e2_l(scen);
29
30 Set dict / scen .scenario.''
31 om1 .randvar .s_om1 
32 om2 .randvar .s_om2 
33 x1 .level .x1_l 
34 x2 .level .x2_l
The line chance E1 E2 0.6 specifies that in at least 60% of all scenarios both E1 and E2 must be satisfied at the same time. There are a total of 12 scenarios, so both constraints must be satisfied in at least 8 (12 * 0.6 = 8) scenarios. We can verify that this requirement has been enforced by checking in the output file the level values of the constraints, i.e. e1,1 and e2,1. Indeed, in the optimal solution both constraints hold in scenarios 4 to 12, so there are 9 scenarios that satisfy both inequalities.

4.3 Individual Chance Constraints

In this section we discuss the general stochastic linear problem with chance constraints assuming that there is no correlation between the probabilities of the rows of the matrix $\tilde{A}$:

$$\begin{align*}
\text{Min} & \quad c^T x \\
\text{s.t.} & \quad P(\tilde{A}_i x \leq \tilde{b}_i) \geq p_i, \quad i = 1, \ldots, m \\
& \quad x \geq 0.
\end{align*}$$

Consider the example from the previous section, this time with individual chance constraints and extended by one constraint:

$$\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad P(\omega_1 x_1 + x_2 \geq 7) \geq 0.75, \quad \omega_1 \in \Omega_1 = \{1, 2, 3, 4\} \\
& \quad P(\omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \quad \omega_2 \in \Omega_2 = \{1, 2, 3\} \\
& \quad P(\omega_1 x_1 + \omega_2 x_2 \geq 10) \geq 0.5, \quad (\omega_1, \omega_2) \in \Omega_1 \times \Omega_2 = \Omega \\
& \quad x_1, x_2 \geq 0.
\end{align*}$$

Note that $\Omega$ is defined as in (23) above, we have again 12 scenarios each with probability $\pi^k = \frac{1}{12}$. However, in this example the first inequality must hold in 9 out of 12 scenarios (0.75 * 12 = 9), the second inequality must hold in 8 out of 12 inequalities (0.6 * 12 = 8) and the third inequality must hold in 6 out of 12 scenarios. Note further that we may have four types of scenarios: scenarios where all constraints are violated, scenarios where two constraints are violated, scenarios where one constraint is violated and scenarios where all three constraints are satisfied. The only condition is that for each constraint there is the respective number of scenarios where the constraint is satisfied. Note further that the random data in the third inequality is a combination of the random data of the first two inequalities. The inequalities for the scenarios are given below:

$$\begin{align*}
k = 1 & : \quad \omega_1^1 = 1, \quad \omega_2^1 = 1 \quad \omega_1^1 x_1 + x_2 \geq 7 \\
& \quad \omega_1^1 x_1 + 3x_2 \geq 12 \\
& \quad \omega_1^1 x_1 + \omega_2^1 x_2 \geq 10 \\
k = 2 & : \quad \omega_1^2 = 1, \quad \omega_2^2 = 2 \quad \omega_1^2 x_1 + x_2 \geq 7 \\
& \quad \omega_1^2 x_1 + 3x_2 \geq 12 \\
& \quad \omega_1^2 x_1 + \omega_2^2 x_2 \geq 10 \\
\vdots \\
k = 12 & : \quad \omega_1^{12} = 4, \quad \omega_2^{12} = 3 \quad \omega_1^{12} x_1 + x_2 \geq 7 \\
& \quad \omega_1^{12} x_1 + 3x_2 \geq 12 \\
& \quad \omega_1^{12} x_1 + \omega_2^{12} x_2 \geq 10
\end{align*}$$
The MIP equivalent follows:

\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 \\
\text{s.t.} & \quad 1x_1 + x_2 \geq 7 - M(1 - y_1^1) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_2^1) \\
& \quad 1x_1 + 1x_2 \geq 10 - M(1 - y_3^1) \\
& \quad 2x_1 + x_2 \geq 7 - M(1 - y_1^2) \\
& \quad 1x_1 + 3x_2 \geq 12 - M(1 - y_2^2) \\
& \quad 2x_1 + 1x_2 \geq 10 - M(1 - y_3^2) \\
& \quad \vdots \\
& \quad 4x_1 + x_2 \geq 7 - M(1 - y_{12}^1) \\
& \quad 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}^2) \\
& \quad 4x_1 + 3x_2 \geq 10 - M(1 - y_{12}^3) \\
cc_1 &= 1 - \sum_k \pi^k y_k^1, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
cc_2 &= 1 - \sum_k \pi^k y_k^2, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
cc_3 &= 1 - \sum_k \pi^k y_k^3, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
x_1, x_2 \geq 0 \\
0 &\leq cc_1 \leq (1 - 0.75) \\
0 &\leq cc_2 \leq (1 - 0.6) \\
0 &\leq cc_3 \leq (1 - 0.5) \\
y_k^j &\in (0, 1).
\end{align*}
\]

As expected, there are three constraints for every scenario. Note that we introduce three new variables, \(cc_1\), \(cc_2\) and \(cc_3\) and three corresponding constraints. Each of the variables has a different range mirroring the different probabilities with which a constraint may be violated.

The core model of the GAMS code is very similar to the core model with joint chance constraints. We just add the third inequality:

\begin{verbatim}
1 Equations OBJ, E1, E2, E3;
2
3 OBJ.. Z =e= X1 + X2;
4 E1.. om1*X1 + X2 =g= 7;
5 E2.. om2*X1 + 3*X2 =g= 12;
6 E3.. om1*X1 + om2*X2 =g= 10;

There is a slight modification in the annotations:

1 file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
2 $onput
3 randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
4 randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
5 chance E1 0.75
6 chance E2 0.6
7 chance E3 0.5
8 $offput
9 putclose emp;
\end{verbatim}

Observe that every constraint is listed separately with its respective probability. Note that in case one constraint has to be satisfied in all scenarios (so it is strictly speaking not a chance constraint), then it has to be listed with probability 1.0.

Here is a summary of which constraints are satisfied in which scenarios in the optimal solution:

| Scenarios where E1 is satisfied: | 4, 5, 6, 7, 8, 9, 10, 11, 12 |
| Scenarios where E2 is satisfied: | 2, 3, 5, 6, 8, 9, 11, 12 |
| Scenarios where E3 is satisfied: | 6, 7, 8, 9, 10, 11, 12 |
Observe that all constraints are satisfied in as many scenarios as required. Note that as predicted there are scenarios where all three constraints are satisfied \( k = 6, 8, 9, 11, 12 \), scenarios where only two constraints are satisfied \( k = 5, 7, 10 \), scenarios where only one constraint is satisfied \( k = 2, 3, 4 \) and one scenario where all constraints are violated \( k = 1 \).

### 4.4 Joint chance constraints vs. individual chance constraints

The choice whether joint or individual chance constraints should be used depends on the system being modeled. Both approaches have their own advantages. Individual chance constraints are weaker since not all constraints have to be satisfied at the same time. This can be clearly observed in the optimal solution for example (22). The objective value is 5.20 in the model with joint chance constraints and 4.75 in the model with individual chance constraints (assuming that each constraint is satisfied in 60% of all scenarios). Since it is a minimizing problem the model with individual chance constraints yields the better result. However, in this solution we have only 6 scenarios where both constraints are simultaneously satisfied while each constraint is satisfied in eight scenarios in total, as required.

### 4.5 Penalizing violations of chance constraints

EMP SP offers the syntax for a penalty factor for each scenario that violates one or more constraints. Taking the joint chance constraints example (22) the emp.info file of the GAMS code could be modified in the following way:

```plaintext
1 file emp / 'emp.info' /; put emp '* problem %gams.i%'/;
2 $onput
3 randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
4 randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
5 chance E1 E2 0.6 3
6 $offput
7 putclose emp;
```

Note that in line 5 we added the penalty factor 3. Recall the MIP equivalent (25) of the problem:

\[
\begin{align*}
\text{Min} & \quad z = x_1 + x_2 \\
\text{s.t.} & \quad 1x_1 + x_2 \geq 7 - M(1−y_1) \\
& \quad 1x_1 + 3x_2 \geq 12 − M(1−y_1) \\
& \quad \vdots \\
& \quad 4x_1 + x_2 \geq 7 - M(1−y_{12}) \\
& \quad 3x_1 + 3x_2 \geq 12 − M(1−y_{12}) \\
cc & = 1 - \sum \pi^k y_k, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& \quad x_1, x_2 \geq 0 \\
& \quad 0 \leq cc \leq (1 - 0.6) \\
& \quad y_k \in (0, 1).
\end{align*}
\]  

(31)

The probability with which the constraints were violated is stored in the variable \( cc \). The introduction of the penalty factor on line 5 of the emp.info file causes \( cc \) multiplied by the penalty factor to be added to the objective function:

\[
z = x_1 + x_2 + 3cc .
\]  

(32)

Similarly, the lines

\begin{verbatim}
chance E1 0.75 5
chance E2 0.6 6
chance E3 0.5 7
\end{verbatim}

in the emp.info file of the GAMS code for the problem with individual chance constraints (26) trigger the objective function in the MIP equivalent to become:

\[
z = x_1 + x_2 + 5cc_1 + 6cc_2 + 7cc_3.
\]  

(33)
This can be useful in order to explore sensitivities to slight changes.
There is also a possibility that allows the modeler to use the probability expression as a variable in the original model. For example, in the joint chance constraints problem above we could introduce a new variable, \( \text{viol} \), in the objective function:

\[
\text{Min } z = x_1 + x_2 + 3 \times \text{viol} \quad (34)
\]

Then we replace line 5 in the emp.info file of the GAMS code as follows:

\text{chance E1 E2 0.6 viol}

This addition causes \( cc_1 \) to be replaced by \( \text{viol} \) in the MIP equivalent. Thus we have:

\[
\text{viol} = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12}, \quad \text{viol} \in [0, 0.4]. \quad (35)
\]

This model is equivalent to the joint chance constraints model with penalty factor 3 with which we started this section.

5 Risk Measures

Risk measures are mechanisms to evaluate the effects of uncertainty in the underlying system on the outcomes of interest. They can be used to modify the distribution of outcomes. In this section we explore how optimization problems involving risk measures can be modeled using stochastic programming in GAMS EMP. Specifically, we examine how an investor might seek to balance expected rewards and the risk of loss when she decides how to allocate assets in a portfolio.

First, we will discuss maximizing the expected value of a portfolio with uncertain returns, then we will introduce the notion of Value at Risk and consider optimization problems involving this risk measure and finally we will examine optimization problems involving Conditional Value at Risk and a combination of expected value and Conditional Value at Risk. Other risk measures could be implemented in future versions of EMP SP. For simplicity of exposition we only describe two-stage models here. In the examples introduced below the period \((0, T)\) is the period between investing in a portfolio of assets and return from this portfolio.

5.1 Expected Value

Suppose an investor has the opportunity to invest a certain amount in three assets. She is given the probability distribution in Table 3 that links each asset with a possible return at time \( T \). The question arises how she should allocate her funds between the three assets at time 0 in order to maximise her expected return at time T.

Table 3: Return by scenario

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Probability</th>
<th>ATT</th>
<th>GMC</th>
<th>USX</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1/12</td>
<td>1.300</td>
<td>1.225</td>
<td>1.149</td>
</tr>
<tr>
<td>s2</td>
<td>1/12</td>
<td>1.103</td>
<td>1.290</td>
<td>1.260</td>
</tr>
<tr>
<td>s3</td>
<td>1/12</td>
<td>1.216</td>
<td>1.216</td>
<td>1.419</td>
</tr>
<tr>
<td>s4</td>
<td>1/12</td>
<td>0.954</td>
<td>0.728</td>
<td>0.922</td>
</tr>
<tr>
<td>s5</td>
<td>1/12</td>
<td>0.929</td>
<td>1.144</td>
<td>1.169</td>
</tr>
<tr>
<td>s6</td>
<td>1/12</td>
<td>1.056</td>
<td>1.107</td>
<td>0.965</td>
</tr>
<tr>
<td>s7</td>
<td>1/12</td>
<td>1.038</td>
<td>1.321</td>
<td>1.133</td>
</tr>
<tr>
<td>s8</td>
<td>1/12</td>
<td>1.089</td>
<td>1.305</td>
<td>1.732</td>
</tr>
<tr>
<td>s9</td>
<td>1/12</td>
<td>1.090</td>
<td>1.195</td>
<td>1.021</td>
</tr>
<tr>
<td>s10</td>
<td>1/12</td>
<td>1.083</td>
<td>1.390</td>
<td>1.131</td>
</tr>
<tr>
<td>s11</td>
<td>1/12</td>
<td>1.035</td>
<td>0.928</td>
<td>1.006</td>
</tr>
<tr>
<td>s12</td>
<td>1/12</td>
<td>1.176</td>
<td>1.715</td>
<td>1.908</td>
</tr>
</tbody>
</table>
Mathematically, the problem can be expressed as follows:

\[
\begin{align*}
\text{Max} & \quad \mathbb{E}[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0, \\
\end{align*}
\]

(36)

where the variable \( R \) is the return (and is a function of the random variable \( v_j \)), \( \mathbb{E}[R] \) the expected return, \( w_j \) the weight associated with each asset \( j \) and \( v_j \) is the (random variable) return of each asset \( j \). The weights can also be interpreted as proportions of the amount to be invested, their sum must be 1. Note that the \( w_j \)'s are the decision variables in this problem.

We present two different ways to model this problem in GAMS using EMP SP. Both models have two stages: in the first stage the weights are chosen without knowing which scenario will be realized, in the second stage the 12 scenarios are taken into account. We start with the part of the code where the data is given. It is named data.inc and incorporated in all models in this section.

1 Set j assets / ATT, GMC, USX /
2 s scenarios / s1*s12 /
3
4 Table vs(s,j) scenario returns from assets
5 att gmc usx
6 s1 1.300 1.225 1.149
7 s2 1.103 1.290 1.260
8 s3 1.216 1.216 1.419
9 s4 0.954 0.728 0.922
10 s5 0.929 1.144 1.169
11 s6 1.056 1.107 0.965
12 s7 1.038 1.321 1.133
13 s8 1.089 1.305 1.732
14 s9 1.090 1.195 1.021
15 s10 1.083 1.390 1.131
16 s11 1.035 0.928 1.006
17 s12 1.176 1.715 1.908;
18
19 Alias (j,jj);
20 Parameter
21 mean(j) mean return
22 dev(s,j) deviations
23 covar(j,jj) covariance matrix of returns
24 totmean total mean return;
25
26 mean(j) = sum(s, vs(s,j))/card(s);
27 dev(s,j) = vs(s,j) - mean(j);
28 covar(j,jj) = sum(s, dev(s,j)*dev(s,jj))/(card(s)-1);
29 totmean = sum(j, mean(j))/card(j);
30 display mean, dev, covar, totmean;
31
32 Parameter
33 p(s) probability / #s [1/card(s)] /
34 v(j) return from assets; v(j) = mean(j);

In the first model we introduce a new variable for the expected return, \( EV_r \), and the new keyword ExpectedValue:

1 $include data.inc
2
3 Variables
4 r value of portfolio under each scenario
5 w(j) portfolio selection
6 EV_r expected value of r
7 objective objective variable;
8 Positive variables w;
9
10 Equations
11 defr return of portfolio
12 budget budget constraint
13 obj.eq objective eqn;
14
15 defr.. r =e= sum(j, v(j)*w(j));
16 budget.. sum(j, w(j)) =e= 1;
17 obj.eq.. objective =e= EV_r;
18 model portfolio / all /;
19
20 file emp / '%emp.info%' /;
21 emp.nd=4;
22 put emp '* problem %gams.i%'

In the first model we introduce a new variable for the expected return, \( EV_r \), and the new keyword ExpectedValue:
In the emp file $\text{EV}_r$ is declared as the ExpectedValue of the random variable $r$, the objective to be maximized is the expected return $\text{EV}_r$. Note that the new variables $\text{EV}_r$ and $\text{obj}$ belong to stage 1. Since the expected value of $r$ is not scenario dependent, its value is known in the preceding stage to the resolution of $r$, namely stage 1. Note that in a 3-stage-problem with $r$ in the third stage, the expected value of $r+$ will be known with certainty in the second stage.

In the second model the fact that the expected return is being maximized is not stated explicitly but is only implied:

Observe that the syntax suggests that the return $r$ is maximized (line 32). However, $r$ is a random variable so in fact the expected return is maximized. Note that the statement emp.nd=4 ensures that 4 decimal places are used for the values of vs in the emp file, the default is 2. Note further that the solver renormalizes the sum of the probabilities to 1 if some input rounding has occurred.

Both models have the same solution. We prefer the first model since the syntax is more explicit and clearer.

### 5.2 Value at Risk (VaR)

Suppose $G(x, \xi)$ is a real valued function of the decision vector $x$ and a random data vector $\xi$ and that it denotes the loss function of a portfolio of assets. We aim to restrict potential losses and so we choose a portfolio composition such that the loss does exceed a certain threshold $\gamma$ ($\gamma \in \mathbb{R}$) with a probability smaller or equal to $\alpha$, $\alpha \in (0, 1)$, where $\alpha$ is small. This condition can be modeled as a chance constraint (compare section Chance Constraints on chance constraints) and has the...
form

\[ P(G(x, \xi) > \gamma) \leq \alpha \]  

(37)

or equivalently,

\[ P(G(x, \xi) - \gamma \leq 0) \geq 1 - \alpha. \]  

(38)

Consider the random variable \( Z_x := G(x, \xi) - \gamma \). For a given value of \( x \), let \( F_Z(z) := P(Z \leq z) \) be the cumulative distribution function of \( Z \). Now, the point \( x \) satisfies the constraint (37) if and only if \( F_Z(0) \geq 1 - \alpha \). This is equivalent to saying that \( x \) satisfies the constraint (38) if and only if \( F_Z^{-1}(1 - \alpha) \leq 0 \).

The (left-side) quantile \( F_Z^{-1}(\theta) \) is called Value at Risk. It is denoted by \( \text{VaR}_\theta(Z) \), i.e.

\[ \text{VaR}_\theta(Z) = \inf \{ t : F_Z(t) \geq \theta \}. \]  

(39)

Hence constraint (38) can be written in the following equivalent form:

\[ \text{VaR}_{1-\alpha}(G(x, \xi)) \leq \gamma. \]  

(40)

Value at Risk as introduced in equation (39) refers to a percentile on the left tail of a distribution. From now on it will be denoted by \( \text{VaR}_\theta(Z) \). When considering the Value at Risk at the right tail of the distribution \( \theta \) typically equals 0.9 or 0.95, it is denoted by \( \text{VaR}_\theta(Z) \).

\[ \text{Figure 32.2: VaR and CVaR} \]

\[ \text{Figure 2 illustrates } \text{VaR}_{0.05}(Z) \text{ where } Z \text{ is normally distributed with mean } \mu = 0.645 \text{ and standard deviation } \sigma = 1. \]
GAMS EMP SP provides the keywords `varlo` and `varup` as a convenient alternative to chance constraints to model Value at Risk. In the example above, the investor might be interested in a strategy that maximizes the threshold at a certain cutoff, say 10% on the left tail of the return curve. Mathematically, the problem can be expressed as follows:

\[
\begin{align*}
\text{Max} & \quad \text{VaR}_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}
\]  

(41)

where \( \text{VaR}_\theta \) is the Value at Risk at the lower \( \theta \)th percentile.

In the emp file of the corresponding GAMS model we introduce a new variable for \( \text{VaR}_r \), a scalar to specify the percentile we are interested in (\( \theta \)) and the keyword `varlo`:

```plaintext
$include data.inc

Scalar
theta relative volume / 0.1 /;

Variables
r value of portfolio under each scenario
w(j) portfolio selection
VaR_r value at risk of r
objective objective variable;

Positive variables w;

Equations
defr return of portfolio
budget budget constraint
obj_eq objective function;

defr.. r =e= sum(j, v(j)*w(j));
budget.. sum(j, w(j)) =e= 1;
obj_eq.. objective =e= VaR_r;

model portfolio / all /;

file emp / '%emp.info%' /
put emp '* problem %gams.i%' /
'varlo r VaR_r ' theta
/stage 2 r defr v'
/stage 1 objective obj_eq VaR_r'
/jrandvar v('att') v('gmc') v('usx')"
loop(s,
put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

Parameter
s_v(s,j) return from assets by scenario /s1.att 1/
s_r(s) return from portfolio by scenario;

Set dict / s .scenario.''

v .randvar. s_v
r .level. s_r /

solve portfolio using emp max objective scenario dict;
```

Note that the objective equals the Value at Risk at the left tail, denoted by \( \text{VaR} \). Line 25 specifies that the variable \( \text{VaR}_r \) is the Value at Risk, \( r \) is the random variable and the scalar \( \theta \) is the percentile (in range 0 to 1) we consider. As in the previous section, the objective, the objective equation and the variable of the objective belong to the first stage while the the equation that handles the random data and all its variables belong to the second stage.

Note that the keyword `varlo` specifies that we are looking at the left tail of the probability distribution. For the right tail of the distribution the keyword to be used is `varup`. The keyword `var` is identical to `varup`. Note further that it is only appropriate to maximize \( \text{VaR}_\alpha \) and minimize \( \text{VaR}_\alpha \).

The implementation of the keywords `varlo` and `varup` is based on a mixed integer program similar to that described in Section Single Chance Constraints. Note that these implementations are likely to be hard and/or time consuming to solve. There is an option that allows the user to customize the big \( M \) value in the same manner that was outlined in Section Single Chance Constraints:

```
$onecho > de.opt
```
Stochastic Programming (SP) with EMP

VaRBigM = 500
$offecho
portfolio.optfile=1;

Note that the default value of $M$ is 1000 and currently only the solver DE supports the keywords for VaR.

In another variation on the example above we consider an investor who aims to take into account both, the expected return and the Value at Risk of the return at a certain threshold $\theta$. She combines the two risk measures and uses a scalar ($\lambda$) as weight. A mathematical formulation of the problem reads as follows:

$$\begin{align*}
\text{Max} & \quad \lambda \mathbb{E}[R] + (1 - \lambda) \text{VaR}_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}$$

(42)

where $\mathbb{E}[R]$ is the expected value of the return and $\text{VaR}_\theta$ is VaR at the $\theta$th percentile.

The GAMS model follows.

1 $include data.inc
2
3 Scalar
4 theta relative volume / 0.1 /;
5 lambda weight EV versus VaR / 0.2 /;
6
7 Variables
8 r value of portfolio under each scenario
9 w(j) portfolio selection
10 VaR value at risk of r
11 EV expected value of r
12 obj objective variable;
13 Positive variables w;
14
15 Equations
16 defr return of portfolio
17 budget budget constraint
18 defobj convex combination for both risk measures;
19
20 defr.. r =e= sum(j, v(j)*w(j));
21 budget.. sum(j, w(j)) =e= 1;
22 defobj.. obj =e= lambda*EV + (1-lambda)*VaR;
23 model portfolio ext / all /;
24
25 file emp / '%emp.info%' /
26 put emp '* problem %gams.i%'
27 / 'ExpectedValue r EV_r'
28 / 'varlo r VaR_r theta'
29 / 'stage 2 r defr v'
30 / 'stage 1 defobj obj'
31 / 'jrandvar v(att) v(gmc) v(usx)'
32 loop(s,
33 put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
34 putclose emp;
35
36 Set dict / s .scenario.''
37 v .randvar. s
38 r .level. s_r /;
39
40 solve portfolio ext using emp max obj scenario dict;

The scalar $\lambda$ is introduced as a weight in the sum in the objective function and in the emp file both keywords ExpectedValue and varlo are used.

Concluding this section we present an alternative way to model VaR. The code is identical to the code of the first model on the model first EV except for the scalar $\theta$ and the emp.info file. The modification of the code for this alternative way of modeling VaR is given below.

1 Scalar
2 theta relative volume / 0.1 /;
3
4 file emp / 'emp.info' /
5 put emp '* problem %gams.i'
Observe that there is no additional variable for VaR, but the risk measure is simply applied to the objective function.

5.3 Conditional Value at Risk (CVaR)

CVaR is the expected average return (in a given time period) given that we are in the \((\alpha \times 100)\%\) left tail of the return distribution, where \(\alpha \in (0, 1)\). In other words, \(CVaR_\alpha\) is a mean of the left tail. For example, if we are interested in the 5\% worst cases, i.e. \(\alpha = 0.05\), \(CVaR_\alpha\) is the conditional expectation of the return, given the return is no greater than VaR.

Let \(\xi\) be a random variable with probability density function \(p(\xi)\), let \(G(\xi)\) be a function of the random variable \(\xi\) denoting the return of a portfolio of assets and let \(\alpha \in (0, 1)\) be a probability. Then the conditional value at risk of \(G(\xi)\) is defined as

\[
CVaR_\alpha(G(\xi)) = \frac{1}{\alpha} \int_{-\infty}^{VaR_\alpha} G(\xi) \cdot p(\xi) d\xi,
\]

where \(VaR_\alpha\) is the value at risk.

In the example above the investor might be interested to make sure that if things get bad she loses as little as possible. She might consider the worst 10\% of possible cases and allocate her funds such that the expected mean return in these cases is maximized. Mathematically, the problem can be expressed as follows:

\[
\begin{align*}
\text{Max} & \quad CVaR_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}
\]

where \(E[R]\) is the expected value of the return and \(CVaR_\theta\) is the CVaR at the confidence level \(\theta\).

This problem can be modeled in GAMS by introducing in the emp file a new variable for the conditional value at risk (CVar_r), a scalar to specify the percentage of the worst cases we are interested in (theta) and cvarlo, a new keyword.

```plaintext
1 $include data.inc
2 3 Scalar
4   theta relative volume / 0.1 /;
5 6 Variables
7   r value of portfolio under each scenario
8   w(j) portfolio selection
9   CVaR_r conditional value at risk of r
10  objective objective variable;
11 Positive variables w;
12 13 Equations
14   defr return of portfolio
15   budget budget constraint
16   obj_eq objective function;
17 18 defr..  r =e= sum(j, v(j)*w(j));
19   budget.. sum(j, w(j)) =e= 1;
20   obj_eq.. objective =e= CVaR_r;
21  model portfolio / all /;
22 23 file emp / '%emp.info%' /
24  put emp '* problem %gams.i%'
25     / 'cvarlo r CVaR_r' theta
26     / 'stage 2 r defr v'
27     / 'stage 1 objective obj_eq CVaR_r'
28     / "jrandvar v('att') v('gmc') v('usx')"
29 loop(s,
30     put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
31 putclose emp;
```
Observe that the objective equals the Conditional Value at Risk (the conditional expectation of the \textit{left} tail, denoted by $CVaR$). Line 25 specifies that the variable $CVaR_r$ is the Conditional Value at Risk, $r$ is the random variable and the scalar $\theta$ is the fraction (in range 0 to 1) we consider. As in the previous section, the objective, the objective equation and the variable of the objective belong to the first stage while the the equation that handles the random data and all its variables belong to the second stage.

Note that the keyword \texttt{cvarlo} specifies that we are looking at the \textit{left} tail of the probability distribution. For the \textit{right} tail of the distribution the keyword to be used is \texttt{cvarup}. The keyword \texttt{cvar} is identical to \texttt{cvarup}. The conditional value of risk denoting the mean of the \textit{right} tail of the distribution can be denoted by $CVaR$ and is defined as:

$$CVaR_\alpha(G(\xi)) = \frac{1}{1-\alpha} \int_{\text{VaR}_\alpha(G(\xi))}^{\infty} G(\xi) \cdot p(\xi) d\xi.$$ \hspace{1cm} (45)

Note that it is only appropriate to maximize $CVaR_\alpha$ and minimize $CVaR_\alpha$. Furthermore, $CVaR_\alpha$ is a concave function and so should only be constrained using e.g. $CVaR_\alpha \geq \gamma$ and $CVaR_\alpha$ is convex, so should only appear in constraints like $CVaR_\alpha \leq \gamma$.

In a final variation on the example above we consider an investor who aims to take into account both, the expected return and the Conditional Value at Risk of the return at a certain threshold $\theta$. She combines the two risk measures and uses a scalar ($\lambda$) to weigh the two summands. A mathematical formulation of the problem reads as follows:

$$\begin{align*}
\text{Max} & \quad \lambda \mathbb{E}[R] + (1-\lambda) CVaR_\theta[R] \\
\text{s.t} & \quad R = \sum_j w_j v_j \\
& \quad \sum_j w_j = 1 \\
& \quad w_j \geq 0,
\end{align*}$$ \hspace{1cm} (46)

where $\mathbb{E}[R]$ is the expected value of the return and $CVaR_\theta$ is the CVaR at the confidence level $\theta$.

The GAMS model follows.
The scalar $\lambda$ is introduced as a weight in the sum in the objective function and in the emp file both keywords ExpectedValue and cvarlo are used. By decreasing the value of $\lambda$ from 1 to 0 we can make the investor increasingly risk averse.

Concluding this section we present an alternative way to model CVaR. The code is identical to the code of the first model on page the model first EV except for the scalar $\theta$ and the emp.info file. The modification of the code for this alternative way of modeling CVaR is given below.

Observe that there is no additional variable for CVaR, but the risk measure is simply applied to the objective function.

6 Summary of keywords and solver configurations

The following keywords can be used in the emp.info file to describe the uncertainty of a problem:

**chance:** This defines individual or joint chance constraints (CC) using the following syntax:

```
chance equ {equ} [holds] minRatio [weight|varName]
```

This way one defines that a single constraint $\text{equ}$ (individual CC) or a set of constraints (joint CC) does only have to hold for a certain ratio ($0 \leq \text{minRatio} \leq 1$) of the possible outcomes. The keyword $\text{holds}$ is optional and does not affect the solver. If $\text{weight}$ is defined, the violation of a CC gets penalized in the objective ($\text{weight} \times \text{violationRatio}$). Alternatively, the violation can be multiplied by an existing variable if this is defined by varName.

**cvarlo:** This keyword assigns a variable to have the value $CVaR_\alpha$.

$\alpha$ is a scalar that represents the confidence level for the Conditional Value at Risk. There are two options for the syntax:

```
cvarlo scalar
```

and

```
cvarlo rv var scalar
```

In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly (rv) and a variable for the value of $CVaR$ is added (var), and scalar is the value of $\alpha$.

**cvarup:** This keyword assigns a variable to have the
value $\text{CVaR}_\alpha$. $\alpha$ is a scalar that represents the confidence level for the Conditional Value at Risk. There are two options for the syntax:

\[
\text{cvarup scalar}
\]

and

\[
\text{cvarup rv var scalar}
\]

In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly (rv) and a variable for the value of CVaR is added (var), and scalar is the value of $\alpha$. Note that the keyword cvar is identical to cvarup which refers to the right tail of the distribution.

**ExpectedValue**: This keyword is used to state that a variable is the expected value of a random variable. The syntax is as follows:

\[
\text{ExpectedValue rv var}
\]

**jrandvar**: Jrandvar can be used to define discrete random variables and their joint distribution:

\[
\text{jrandvar rv rv \{rv\} prob val val \{val\} \{prob val val \{val\}}
\]

At least two random variables rv are defined and the outcome of those is coupled. The probability of the outcomes is defined by prob and the corresponding realization for each random variable by val.

**randvar**: This defines both discrete and parametric random variables:

\[
\text{randvar rv discrete prob val \{prob val\}}
\]

The distribution of discrete random variables is defined by pairs of the probability prob of an outcome and the corresponding realization val.

\[
\text{randvar rv distr par \{par\}}
\]

A list of all supported parametric distributions can be found in Table Table 2. All possible values for distr and the related parameters par are listed there.

**sample**: This allows the user to customize the size of the sample of a random variable from a continuous distribution and there is also the option to determine the variance reduction method to be used:

\[
\text{sample rv1 \[rv2 ... rvn\] sampleSize \[varRedMethod\]}
\]

In rv the name of the random variable is given. The sample size of more than one random variable can be customized simultaneously. In this case the names of the random variables are listed. sampleSize is a number, namely the desired size of the sample and varRedGroup is optional. For more details about variance reduction methods see section Sampling and in addition, please consult the LINDO manual.

**setSeed**: This sets the seed for the random number generator of the sampling routines called using the sample keyword. If setSeed is used in the emp.info file, the seed is set once before we generate all samples.

\[
\text{setSeed seed}
\]

**stage**: Random variables (rv), equations (equ) and variables (var) are assigned to non-default stages like this:

\[
\text{stage stageNo rv \| equ \| var \{rv \| equ \| var\}}
\]

StageNo defines the stage number. The default stage for all random variables, equations and variables not mentioned with the stage keyword is 1, except for the objective equation. The default for the objective equation is the highest stage mentioned. Note that the objective variable is in stage 1.

**varlo**: This keyword assigns a variable to have the value $\text{VaR}_\alpha$.

where $\alpha$ is a scalar that represents the percentile of the Value at Risk. There are two options for the syntax:

\[
\text{varlo scalar}
\]

and

\[
\text{varlo rv var scalar}
\]
In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly \texttt{rv} and a variable for the value of VaR is added (\texttt{var}), and the scalar is the value of \(\alpha\).

\textbf{varup}: This keyword assigns a variable to have the value \(VaR_\alpha\).

where \(\alpha\) is a scalar that represents the percentile of the Value at Risk. For \(VaR\ \alpha\) typically equals 0.95 or 0.9. There are two options for the syntax:

\begin{verbatim}
varup scalar and varup rv var scalar
\end{verbatim}

In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly (\texttt{rv}) and a variable for the value of VaR is added (\texttt{var}), and the scalar is the value of \(\alpha\). Note that the keyword \texttt{var} is identical to \texttt{varup} which refers to the right tail of the distribution.

At the moment four GAMS solvers can be used to solve SP models in the way described in this document: DE, DECIS, LINDO and JAMS. Further information about these solvers can be found in the corresponding solver manuals.

Not all keywords mentioned above are supported by all four solvers. The following table specifies which keywords can be used with which solvers. The keywords not mentioned in the table are supported by all four solvers.

\textbf{Table 4: Solver Capabilities}

<table>
<thead>
<tr>
<th></th>
<th>DE</th>
<th>DECIS</th>
<th>LINDO</th>
<th>JAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>chance</td>
<td></td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jrandvar</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>randvar (discrete)</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>randvar (parametric)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sample</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>setSeed</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>var</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvvar</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>ExpectedValue</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

The SP options available for the LINDO solver are documented in the "LINDO/LINDOGlobal manual".

7 More on scenarios and output extraction

The size of the set \texttt{scen} does not have to match the number of scenarios actually generated in the solution process. For example, if we set the size of \texttt{scen} to 2 in the news vendor model from section \textbf{Uncertain demand: discrete distribution}, then the results of the first two scenarios will be stored in the parameters \texttt{s_d} , \texttt{s_x} and \texttt{s_s}, and the results of the other scenarios will not be stored. On the other hand, if the size of \texttt{scen} is bigger than the number of scenarios generated, then in the parameters (e.g. \texttt{s_d}) the positions of the surplus elements of \texttt{scen} will be empty.

After solving a SP model only the solution of the expected value problem can be accessed via the regular .L and .M fields. As described in section \textbf{Uncertain demand: discrete distribution}, additional parameters have to be defined to store the results for the different scenarios solved. The following values can be stored by this approach:

\begin{verbatim}
level: Stores the levels of a scenario solution of variable or equation.
Marginal: Stores the marginals of a scenario solution of variable or equation.
randvar: Stores the realization of a random variable.
opt: Stores the probability of each scenario.
\end{verbatim}

In the news vendor model from section \textbf{Uncertain demand: discrete distribution} we can use the following:

\begin{verbatim}
Set scen Scenarios / s1*s6 /;
\end{verbatim}
Parameter

s_x(scen) Units bought by scenario
s_rep(scen,*) Scenario probability / #scen.prob 0/;

Set dict / scen .scenario.''
  x .level .s_x
  , . opt .s_rep/;

The size of the set scen defines the number of scenarios we are willing to store results for. X is a variable for which we want to access the level and s_x is the parameter the levels of x are stored in. Note that s_x needs to have the same indices as x plus the additional index scen in the first position. In the parameter s_rep we store the probabilities of the different scenarios solved.
EXAMINER

Contents

1 Introduction ......................................................... 749
2 Usage ............................................................... 749
  2.1 Solution Points: Definition .................................. 750
  2.2 Checks Performed .......................................... 750
  2.3 Scaling ....................................................... 751
3 Options ............................................................. 751
  3.1 General Options ............................................. 751
  3.2 Tolerance Options .......................................... 752

1 Introduction

This document describes GAMS/Examiner, a tool for examining points and making an unbiased, independent assessment of their merit. In short, it checks if solutions are really solutions. As an example, it can take a solution point reported as optimal by a solver and examine it for primal feasibility, dual feasibility, and optimality. Examiner has a number of different modes, allowing it to check the input point from GAMS/Base as well as the solution passed by a solver back to GAMS.

Many of the tests done by Examiner (perhaps all of them) are already being done by the GAMS solvers, so Examiner is in a sense redundant. However, the ability to make an independent, transparent check of a solver’s solution is very useful in solver development, testing, and debugging. It is also useful when comparing the solutions returned by two different solvers. Finally, a tool like the Examiner allows one to examine solutions using different optimality tolerances and optimality criteria in a way that is not possible when working with the solvers directly.

GAMS/Examiner is installed automatically with your GAMS system. Without a GAMS/Base license, examiner will run in student or demonstration mode (i.e. it will examine small models only).

2 Usage

Examiner can be used with all supported model types. Since Examiner doesn’t really solve any problems, it is not a good choice for a default solver, and when installing GAMS it does not appear as an option in the list of possible solver defaults. However, you can choose Examiner via the command line:

    gams transport LP=examiner;

or via a GAMS option statement

    option LP=examiner;

somewhere before the solve statement.
Since Examiner is not really a solver, many of the usual GAMS options controlling solvers have no impact on it. However, the `sysout` option is interpreted in the usual way.

The optimality checks done in Examiner are first-order optimality checks done at a given point. A discussion here of these conditions and all they imply would be redundant: any good intro text in optimization will cover them. For linear programming, first-order optimality is all one needs to prove global optimality. For nonlinear programming, these conditions may or may not be necessary or sufficient for optimality; this depends on the convexity of the feasible set and objective and the form of the constraints. For integer programming models, these checks only make sense if we turn the global problem into a local one by adding bounds to the model, essentially fixing each discrete variable to its current value: these bounds are added automatically by Examiner.

Examiner runs in two basic modes of operation: it can examine the input point passed from GAMS/Base to the solver, and it can examine the point passed from the solver back to GAMS. Each mode can be used independent of the other. By default it will operate in the first mode, examining the initial solution passed to it by GAMS, but only if GAMS indicates it is passing an advanced basis to the solver (cf. the GAMS User Guide and the `bratio` option). If you wish to use the second solver-check mode, you may specify an appropriate subsolver using the `subsolver` option (see section Options). If no subsolver is selected, the default solver for the model type being solved is used. In most cases you will want to use an option file to specify exactly what type of examination you wish to perform. The rules for using an option file are described in Chapter 1, "Basic Solver Usage".

### 2.1 Solution Points: Definition

There are a number of different ways a solution point can be defined. Of course the different definitions will typically result in the same points being produced, but there are cases where this will not be precisely so. Since Examiner is intended to explore and analyze these cases, we must make these definitions precise. The following four points are defined and used in Examiner:

1. **The `gamspoint`** is the input point provided by GAMS to Examiner. The GAMS input point includes level & marginal values for the rows and columns: Examiner uses these exactly as given.

2. **The `initpoint`** is determined by the variable levels (primal vars) and equation marginals (dual vars) provided by GAMS to Examiner. These values are used to `compute` the equation levels and variable marginals / reduced costs, using the function evaluator in Examiner rather than the values passed in by GAMS.

3. **The `solupoint`** is similar to the `initpoint`: it uses the variable levels (primal vars) and equation marginals (dual vars) to `compute` the equation levels and variable marginals. The variable levels and equation marginals used are those returned by the subsolver.

4. **The `solvpoint`** is the point returned by the subsolver. The subsolver returns both level and marginal values for the rows and columns: Examiner uses these, exactly as given.

### 2.2 Checks Performed

There are a number of checks that can be performed on any of the solution points. By default, Examiner tries to choose the appropriate checks. For example, if a primal simplex solver returns a models status of nonoptimal, the only checks that makes sense are feasibility in the primal variables and constraints. However, this automatic choice of appropriate checks is not possible when checking points passed in from GAMS/Base.

1. **Primal variable feasibility**: check that all primal variables are within bounds.

2. **Primal constraint feasibility**: check that all primal constraints are satisfied.

3. **Dual variable feasibility**: check that all dual variables are within bounds.

4. **Dual constraint feasibility**: check that all dual constraints are satisfied.

5. **Primal complementary slackness**: check complementarity between the primal variables and the dual constraints / reduced costs.
6. **Dual complementary slackness**: check complementarity between the dual variables / equation marginals, and equation slacks.

7. **Equilibrium condition complementarity**: check complementarity of the equation/variable pairs in complementarity models (MCP, MPEC).

The checks above are implemented with default tolerances. These tolerances can be changed via an option file (see section *Options*).

Different ways exist to check the items mentioned above. For example, different norms can be used to measure the error of the residual when checking for primal feasibility. Currently, we have only implemented one way to make these checks.

### 2.3 Scaling

By default, Examiner makes its checks on the original, unscaled model. In many cases, however, it is important to take scaling into account. Consider the effect of row scaling on the simple constraint \(x^2 \leq 9\) where \(x = 3.5\). Multiplying this constraint through by large or small constants changes the amount of the constraint violation proportionately, but the distance to feasibility is not changed. Applying row scaling to the original model eliminates this problem.

Most solvers scale a model before solving it, so any feasibility or optimality checks and tolerances are applied to the scaled model. The process of unscaling the model can result in a loss of feasibility or optimality. Even though we do not have access to the scales applied by the solver and cannot precisely construct the same scaled model, we can get a better idea of how the solver performed by looking at a model scaled by Examiner rather than by looking at the original.

It is also interesting to see what the model scaling looks like, even if we do not apply the scales to do the Examiner checks. If the row scales are in a nice range, say \([1,100]\), we can have some confidence that the model is well-scaled. In contrast, if the row scales are in the range \([1,1e8]\) we may question the precision of the solution provided.

For each row, Examiner computes the true row scale as

\[ \max(\|RHS_i\|, \max_j(\|A_{ij}\| : \max(1,\|x_j\|))) \]

In this way variables with a large level value lead to large scale factors. Using an option line of `AbsXScale 0` makes the scale factor independent of the variable values. This replaces the term \(\max(1,\|x_j\|)\) above with 1.

Since the user may wish to limit the size of the scale factors applied, the true row scales are projected onto the scale factor bounds to get the applied scale factors. The scale factors are applied when making a scaled check by dividing the rows by the scale factors and multiplying the corresponding Lagrange multipliers by these same factors. When making unscaled checks information about the true scales is still included in the output to give the user a hint about potential scaling issues.

Note that the scaled and unscaled checks are made independently. By default only the unscaled checks are performed. If you turn the scaled checks on via an option file line `scaled 1`, this will not turn off the unscaled checks. You will need an option file line of `unscaled 0` to turn off unscaled checks.

### 3 Options

The following options control the behavior of GAMS/Examiner. Many of these are boolean (i.e. on/off) options. In this case, zero indicates off, nonzero on. For details on how to create and use an option file, see the introductory chapter on solver usage.

#### 3.1 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>absXScale</td>
<td>Whether to make scale factors dependent on x values. If on, the matrix coefficients are multiplied by (\max(1,\text{abs}(x))) when computing the scale factors. If off, the matrix coefficients are taken as is. See Section <em>Scaling</em>.</td>
<td>1</td>
</tr>
</tbody>
</table>
### 3.2 Tolerance Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dualCSTol</td>
<td>Tolerance on dual complementary slackness, i.e. between the dual variables and the primal constraints.</td>
<td>1e-7</td>
</tr>
<tr>
<td>dualFeasTol</td>
<td>Tolerance on dual feasibility, i.e. to check feasibility of the dual variables and the dual constraints.</td>
<td>1e-6</td>
</tr>
<tr>
<td>ECTol</td>
<td>Tolerance on equilibrium condition complementarity. Applicable to MCP and MPEC models, where the equilibrium conditions are given by the equation-variable pairs in the model statement.</td>
<td>1e-6</td>
</tr>
</tbody>
</table>

---

**Option**

- **dumpGamsPoint**: Whether to dump the GamsPoint to a basis file in GAMS source format. 0
- **dumpInitPoint**: Whether to dump the InitPoint to a basis file in GAMS source format. 0
- **dumpSoluPoint**: Whether to dump the SoluPoint to a basis file in GAMS source format. 0
- **dumpSolvPoint**: Whether to dump the SolvPoint to a basis file in GAMS source format. 0
- **examineGamsPoint**: Whether to examine the GamsPoint. 0
- **examineInitPoint**: Whether to examine the InitPoint. By default, this option is on if GAMS/Base passes an advanced basis, and off otherwise. auto
- **examineSoluPoint**: Whether to examine the SoluPoint. By default, this option is on if a subsolver has been selected, and off otherwise. auto
- **examineSolvPoint**: Whether to examine the SolvPoint. By default, this option is on if a subsolver has been selected, and off otherwise. auto
- **fCheckAll**: If set, forces all checks on or off. auto
- **fCheckATTR**: If set, forces the model attributes check on or off. auto
- **fCheckDCMP**: If set, forces the dual complementary slackness check on or off. auto
- **fCheckDCON**: If set, forces the dual constraint feasibility check on or off. auto
- **fCheckDVAR**: If set, forces the dual variable feasibility check on or off. auto
- **fCheckPCMP**: If set, forces the primal complementary slackness check on or off. auto
- **fCheckPCON**: If set, forces the primal constraint feasibility check on or off. auto
- **fCheckPVAR**: If set, forces the primal variable feasibility check on or off. auto
- **perpSys**: Controls output during examination of solution points. If on, print out the point in a way that allows for easy visual inspection and verification of the KKT or first order optimality conditions. First, the primal level values and bounds are printed next to the reduced costs. Next, the duals levels and bounds are printed next to the row slacks. 0
- **returnGamsPoint**: Whether to return the GamsPoint as a solution to GAMS/Base. 0
- **returnInitPoint**: Whether to return the InitPoint as a solution to GAMS/Base. auto
- **returnSoluPoint**: Whether to return the SoluPoint as a solution to GAMS/Base. auto
- **returnSolvPoint**: Whether to return the SolvPoint as a solution to GAMS/Base. auto
- **scaled**: Whether to apply checks to a scaled version of the model. 0
- **scaleLB**: Lower bound for applied row scales. 1
- **scaleUB**: Upper bound for applied row scales. maxdouble
- **subSolver**: Indicates what subsolver to run. By default, the subsolver used is the default subsolver for the model type in question. auto
- **trace**: If set, trace information will be computed and appended to this file. auto
- **unScaled**: Whether to apply checks to the original, unscaled version of the model. 1
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>primalCSTol</td>
<td>Tolerance on primal complementary slackness, i.e. between the primal variables and the dual constraints.</td>
<td>$1e^{-7}$</td>
</tr>
<tr>
<td>primalFeasTol</td>
<td>Tolerance on primal feasibility, i.e. to check feasibility of the primal variables and constraints.</td>
<td>$1e^{-6}$</td>
</tr>
<tr>
<td>showTol</td>
<td>relative tolerance for showSlacks - we only want to see explicit slacks that are relatively small</td>
<td>$1e^{-4}$</td>
</tr>
</tbody>
</table>
1 Introduction

GAMS/AMPL allows users to solve GAMS models using solvers within the AMPL modeling system. The GAMS/AMPL link comes free with any GAMS system. Users must have a licensed AMPL system installed and have the AMPL executable in their path.

To run GAMS/AMPL, just specify the solver as ampl. For example, if we wish to solve the transport.gms model, we would run

>> gams transport.gms lp=ampl

For other GAMS solvers, options can be passed on via solver option files. GAMS/AMPL specific options are described in the section GAMS/AMPL Options.

By default, GAMS/AMPL returns a model status of 14 (no solution) and a solver return status of 1 (normal completion), provided the link is executed normally. This includes the case where the AMPL executable is not found.

GAMS external equations and extrinsic functions cannot be used with GAMS/AMPL.

2 AMPL Path

GAMS searches for an AMPL executable using the following hierarchy:

- Via the options AmplPath and RunAmpl within a GAMS/AMPL solver option file.
- An amplpath.txt file located in the GAMS system directory specifying the path of the AMPL executable.
- The system path.

For example, GAMS will first search for the AMPL executable within the ampl.opt file, if specified. If not found, it will search within the GAMS system directory for a file called amplpath.txt specifying the AMPL directory. If amplpath.txt is still not found GAMS will try the system path.

If no AMPL executable is found, the user will see a message similar to
There may also be an output indicating that AMPL was not found, either because it was not installed or because it was not found in the system path.

### 3 GAMS/AMPL Options

#### 3.1 General options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmplPath</td>
<td>Path to AMPL System files</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Verbation AMPL options</td>
<td></td>
</tr>
<tr>
<td>RunAmpl</td>
<td>Name of AMPL executable</td>
<td></td>
</tr>
<tr>
<td>TolNone</td>
<td>Tolerance to interpret status none</td>
<td>1e-12</td>
</tr>
</tbody>
</table>

#### 3.2 Other options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DotMod</td>
<td>AMPL input file name</td>
<td>ampl.mod</td>
</tr>
</tbody>
</table>

The `Option` specifier is used to specify desired AMPL options within the AMPL modeling system. For example, if a user wishes to run AMPL/MINOS with the options `timing=3` `outlev=2` then the user creates a file called `ampl.opt` with the entry

```plaintext
option minos_options "timing=3 outlev=2";
```
# GAMSCHK

Abridged GAMSCHK USER DOCUMENTATION - Version 1.1
A System for Examining the Structure and Solution Properties of Linear Programming Problems Solved using GAMS
Bruce A. McCarl, Professor, Department of Agricultural Economics, Texas A&M University

April 2013 :

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  GAMSCHK USER DOCUMENTATION</td>
<td>758</td>
</tr>
<tr>
<td>2  General Notes on Package Usage</td>
<td>758</td>
</tr>
<tr>
<td>2.1 Selecting a Procedure and Providing Input - the *.GCK File</td>
<td>759</td>
</tr>
<tr>
<td>2.2 The *.GCK file: General Notes on Item Selection</td>
<td>759</td>
</tr>
<tr>
<td>2.3 Procedure Output</td>
<td>761</td>
</tr>
<tr>
<td>2.4 Nonlinear Terms</td>
<td>761</td>
</tr>
<tr>
<td>2.5 Entering Comments in the *.GCK File</td>
<td>761</td>
</tr>
<tr>
<td>2.6 Controlling Page Width in the *.GCK File</td>
<td>761</td>
</tr>
<tr>
<td>2.7 Running Multiple Procedures</td>
<td>761</td>
</tr>
<tr>
<td>3  Use of the Procedures</td>
<td>761</td>
</tr>
<tr>
<td>3.1 DISPLAYCR</td>
<td>761</td>
</tr>
<tr>
<td>3.2 MATCHIT</td>
<td>762</td>
</tr>
<tr>
<td>3.3 ANALYSIS</td>
<td>763</td>
</tr>
<tr>
<td>3.4 BLOCKLIST</td>
<td>763</td>
</tr>
<tr>
<td>3.5 BLOCKPIC</td>
<td>764</td>
</tr>
<tr>
<td>3.6 PICTURE</td>
<td>764</td>
</tr>
<tr>
<td>3.7 POSTOPT</td>
<td>765</td>
</tr>
<tr>
<td>3.8 ADVISORY</td>
<td>765</td>
</tr>
<tr>
<td>3.9 NONOPT</td>
<td>766</td>
</tr>
<tr>
<td>4  Options File</td>
<td>766</td>
</tr>
<tr>
<td>4.1 Solver Choice Options</td>
<td>766</td>
</tr>
<tr>
<td>4.2 When Should I Use SOLVE or NOSOLVE</td>
<td>767</td>
</tr>
<tr>
<td>4.3 Control of Number of Variable and Row Selections Allowed</td>
<td>767</td>
</tr>
<tr>
<td>4.4 Scaling</td>
<td>767</td>
</tr>
<tr>
<td>4.5 NONOPT Filters</td>
<td>767</td>
</tr>
<tr>
<td>4.6 Example Options File</td>
<td>768</td>
</tr>
<tr>
<td>4.7 Solver Options File</td>
<td>768</td>
</tr>
<tr>
<td>5  Known Bugs</td>
<td>768</td>
</tr>
<tr>
<td>6  Tables</td>
<td>769</td>
</tr>
<tr>
<td>7  Appendix A: Reserved Names</td>
<td>773</td>
</tr>
<tr>
<td>8  Appendix B: GAMSCHK One Page Summary</td>
<td>774</td>
</tr>
<tr>
<td>9  Appendix C: Summary of GAMSCHK Options</td>
<td>775</td>
</tr>
<tr>
<td>10 GAMSCHK References</td>
<td>775</td>
</tr>
</tbody>
</table>
1 GAMSCHK USER DOCUMENTATION

This document describes procedures designed to aid users who wish to examine empirical GAMS models for possible flaws. The conceptual basis for many of the routines herein is supplied in McCarl and Spreen, and McCarl et.al. This package of routines is designed for use on any GAMS platform, but for now is implemented on the HP, PC, DEC Alpha, IBM RS6000 and SUN workstations. The function of the specific components of GAMSCHK are to:

- List coefficients for user selected equations and/or variables using the DISPLAYCR procedure.
- List the characteristics of selected groups of variables and/or equations using MATCHIT.
- List the characteristics of equation and variable blocks using BLOCKLIST.
- Examine a GAMS model to see whether any variables and equations contain specification errors using ANALYSIS.
- Generate schematics depicting the characteristics of coefficients by variable and equation blocks using BLOCKPIC.
- Generate a schematic for small GAMS models or portions of larger models depicting the location of coefficients by sign and magnitude using PICTURE.
- Reconstruct the reduced cost of variables and the activity within equations after a model solution using POSTOPT.
- Help resolving problems with unbounded or infeasible models using NONOPT and ADVISORY.

2 General Notes on Package Usage

GAMSCHK must replace a solver. This is done using a GAMS option statement of the form:

```gams
OPTION LP= GAMSCHK;
```

or

```gams
OPTION NLP=GAMSCHK;
```

or

```gams
OPTION MIP=GAMSCHK;
```

which replaces either the NLP, LP, or MIP solver with GAMSCHK. In turn, the user will invoke the solver using the statement:

```gams
SOLVE MODELNAME USING LP MINIMIZING OBJNAME;
```

where MODELNAME is the name used in the GAMS MODEL statement; OBJNAME is the objective function name for the model; and the type of solver that GAMSCHK has replaced which must also be able to solve this type of problem (LP, NLP, or MIP) is identified. The following are examples of GAMS sequences which can be added to the GAMS file:

```gams
OPTION NLP=GAMSCHK;
SOLVE TRANSPORT USING NLP MINIMIZING Z;
```

or

```gams
OPTION LP=GAMSCHK;
SOLVE FEED USING LP MINIMIZING COST;
```

or

```gams
OPTION MIP=GAMSCHK;
SOLVE RESOURCE USING MIP MAXIMIZING PROFIT;
```

1 In all cases, users will be able to replace the LP solver. Replacement of the other solvers depends on the solver licenses owned by the user.
2.1 Selecting a Procedure and Providing Input - the *.GCK File

GAMSCHK requires that the user indicate which procedures are to be employed. This is specified through the use of the *.GCK file where the * refers to the filename from the GAMS execution instruction. The general form of that file is:

```
FIRST PROCEDURE NAME
   ITEM SELECTION INPUT

SECOND PROCEDURE NAME
   ITEM SELECTION INPUT
```

Spaces and capitalization are ignored in this input. For example, a *.GCK file could look like

```
DISPLAYCR
   variables
      SELL(*,*,FANCY)
   maketable
Invariables
   transport(plant2,*,fancy)
Equations
   objT
   notthere
inequations
   resourceq(plant1)
```

PICTURE

The first procedure name in this case is DISPLAYCR and the following 10 lines indicate the items to be selected. Then, we also request PICTURE. Selection entries are treated using several assumptions. In particular:

1. If the *.GCK file is empty then it is assumed that the BLOCKPIC procedure is selected.
2. Spaces may be freely used in the GCK input file.
3. Upper, lower, or mixed case input is accepted.
4. GAMSCHK recognizes certain words. These words are listed in Appendix A and cannot be used as variable or equation names.

2.2 The *.GCK File: General Notes on Item Selection

Some of the procedures permit selection of variables, equations or functions. Specifically, the DISPLAYCR, PICTURE, POSTOPT, and MATCHIT procedures accept input identifying the variables and equations to be utilized. Also NONOPT accepts limited input controlling its function. General observations about the selection requests are

1. Variables can be chosen by entering the word VARIABLE or VARIABLES possibly with a modifier, followed by variable selection statements.
2. Variables can also be selected using the INEQUALATION or INEQUALATIONS syntax followed by names of equations. Use of this syntax results in selection of variables with coefficients in the named equations.
3. Equations are selected by entering the keyword, EQUATION or EQUATIONS possibly with a modifier, followed by equation selection statements.

---

2Thus, if the GAMS instructions are in the file called MYMODEL, and GAMS is invoked using the DOS command GAMS MYMODEL, then the GCK file would be called MYMODEL.GCK. If GAMS instructions are on the filename with a period in it then the name up to the period will be used, i.e., the GCK file associated with MYMODEL.IT would be MYMODEL.GCK
4. Equations can also be selected using the IN_VARIABLE or IN_VARIABLES syntax followed by names of variables. Use of this syntax results in selection of equations in which the named variables have coefficients.

5. Certain item selection modifier keywords can be used depending on procedure. The INTERSECT keyword works with procedures DISPLAYCR and POSTOPT. The INEQUALITY and IN_VARIABLE keywords work with procedures DISPLAYCR, PICTURE and POSTOPT. LISTEQUATION and LIST_VARIABLE keywords work with the MATCHIT procedure. INSOLUTION, NOTINSOLUTION, BINDING, and NOTBINDING keywords work with POSTOPT. The keywords VERBOSE and IDENTIFY work with NONOPT.

6. If variable or equation names do not follow the keyword, then usually all variables or equations are assumed selected.

When variables or equations are to be selected after an item selection keyword, a number of input conventions apply. These conventions are:

1. If a variable or equation name is entered without any following parentheses, then all cases for that variable or equation are selected.

2. The selection entries identify specific elements from among the sets over which the variables and equations are defined. In specifying these elements one can use various wild card entries as discussed below or an element name. Note GAMS set or subset names cannot be used. Set membership information is not available to the GAMSCHK routines.

3. Wild cards can be used to select items. An "*" will select any item. For example, "B*" will select anything starting with a B. "A?B" will select anything beginning with A, ending with B with one intervening alpha numeric character.

4. When individual elements are specified, you need not enclose them in quotes (").

5. Quotes must be specified to include set item names with spaces, and special characters. In that case wild cards do not work and all input up to the next quote is simply copied.

6. When the selected item has more dimensions than specified, then all later dimensions are handled as if a wild card were specified. For example, when a variable X is defined with reference to 4 sets in the GAMS instructions, but only 3 parameters are specified in the GAMSCHK input, then the request is handled as if all elements of the 4th are desired.

7. When the selected item has less dimensions in GAMS than in the item selection input, then all additional dimensions are ignored. Thus, when a variable X is defined with reference to 3 sets in GAMS, but 4 parameters are specified in the item selection file, then the 4th specification is ignored.

8. Multiple selection statements can appear on successive lines of the *.GCK file. Output is ordered according to the way items are found in the GAMS file which is determined by the ordering of variables, equations, and set elements in the original GAMS input.

9. Error messages will be generated when an entry cannot be matched to a GAMS element.

10. Examples include

<table>
<thead>
<tr>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>X(*,CLEVELAND)</td>
<td>which indicates that X will be selected for any element of the first set where the element in the second set equals CLEVELAND</td>
</tr>
<tr>
<td>X(SEATTLE)</td>
<td>when X is two dimensional selects all cases where the first set element is SEATTLE</td>
</tr>
<tr>
<td>X(SEATTLE,CHICAGO,Z)</td>
<td>when X is two dimensional selects the case where the first set element equals SEATTLE, and the second element equals CHICAGO. The third is ignored.</td>
</tr>
<tr>
<td>X</td>
<td>all X's will be selected</td>
</tr>
<tr>
<td>X(S*, C.O, Z)</td>
<td>when X is three dimensional selects where all X's with first element starting with S, second element beginning with C and ending with O and third element Z will be selected.</td>
</tr>
<tr>
<td>*</td>
<td>all variables or equations will be selected</td>
</tr>
<tr>
<td>{empty selection set}</td>
<td>all variables or equations will be selected</td>
</tr>
</tbody>
</table>
2.3 Procedure Output

In all cases the output generated by the procedure will be written to the *.LST file associated with the GAMS call. Thus, if the file is called MODEL with the *.GCK file (MODEL.GCK), then all output will be on MODEL.LST.

2.4 Nonlinear Terms

GAMS models examined with GAMSCHK may involve nonlinear terms. In such cases, GAMSCHK uses the value of the nonlinear term sent forth from GAMS which is an accurate marginal, not total value. GAMS develops this value based on the current level value of the variable. This will either be: a) the starting point selected by GAMS, if the model has not been solved, or b) the current solution value, if the model has been solved. The most accurate portrayals of the coefficients will be generated after the model has been solved through a GAMS SOLVE command before invoking GAMSCHK. Some cases may require a solution and/or the specification of a good starting point before using GAMSCHK. Also, nonlinear terms potentially cause misleading coefficients as those values are local marginal, not global, values determined by the current levels of the variables. Nonlinear terms are marked with *** in the DISPLAYCR, POSTOPT, and NONOPT output.

2.5 Entering Comments in the *.GCK File

The *.GCK file has been programmed so that users can enter comments. These comments can take one of two forms. Comments that begin with a hash mark are copied to the output when the program runs. Comments which begin with a question mark are simply overlooked. Thus, one can temporarily comment GAMSCHK selection statements making them inactive by putting in question marks. If multiple procedures are being run or if some sort of output is decided to screen in the computer output then the hash marks can be entered.

2.6 Controlling Page Width in the *.GCK File

When running multiple procedures, in particular the pictures with other procedures, it is often desirable to have some procedures run with wide page widths, but the rest with a narrower page width. The GCK file provides the option to narrow the page width using a PW= command. In particular, what one can do is run GAMS with a large page width, i.e. run GAMS BLOCK pw=200, then insert in the GCK file instructions which narrow that page width for selected procedures. Users should note that the page width can never be made any wider than the default page width when running with GAMS. Information in excess of the page width will be ignored. Thus, if the model is run under the default status which has a page width of 75 characters then GAMSCHK will reduce the page width down to the maximum page width allowed. Consequently, the pw= command can only be used to narrow the page width from the default page width, not increase it.

2.7 Running Multiple Procedures

GAMSCHK can run multiple procedures during one job. This is done by simply stacking the sequence of the commands in the .GCK file.

3 Use of the Procedures

The following section describes the procedures available in GAMSCHK and their input requirements.

3.1 DISPLAYCR

Brief Purpose: DISPLAYCR displays all coefficients from the empirical model for a set of user selected equations and variables. All nonzero coefficients under each selected variable or in each selected equation are displayed with the associated variable or equation name and coefficient value. The selection entries may refer to all terms in equations under variables or only those coefficients at the intersection of the selected variables and equations.
Usage Notes: This option mirrors the GAMS LIMCOL and LIMROW options, but allows the user to select the specific items to be displayed. Partial displays within a variable or equation are also allowed using INTERSECT. Use of VARIABLE and EQUATION keywords followed by selection statements allows one to select variables and equations. Use of the INvariable command allows users to select the equations which are associated with a particular variable. For example, if one is having trouble with a particular variable and wants to look at competition in the equations in which it appears, then selecting the variable under the INvariable command will display the complete contents of all the equations in which the selected variables have coefficients. Similarly, the INequation command will display the complete contents of all variables which fall in a particular equation. Nonlinear terms are marked with ∗∗∗. When the keyword INTERSECT is found then only the coefficients at the intersection of the specified equations and variables are selected. Use of INTERSECT with the INvariable syntax results in the named variables and the equations in which they fall being selected. Similarly, use of INTERSECT with the INequation syntax results in selection of the named equations and the variables which fall in those equations.

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DEscale feature of the solver options file.

Input File: The keyword DISPLAYCR is entered followed by optional lines of item selection input identifying the variables and equations to be displayed. This file can contain the keywords VARIABLE, INvariable, EQUATION, and INequation, with each followed by a specification of the items to be selected using the procedure input specification conventions that were described above. The keyword INTERSECT can also be used. Several special cases are relevant:

- If none of the above keywords are found after DISPLAYCR and another procedure name does not follow, then the input is assumed to identify variables.
- If input is found but the VARIABLE or INequation keyword cannot be found then no variables are assumed selected.
- If the VARIABLE keyword is entered, but is followed by the end of file or an Appendix A reserved word and INequation does not appear, then all variables are assumed selected.
- If the EQUATION or INvariable keyword cannot be found, then no equations are assumed selected.
- If the EQUATION keyword is entered, but is followed by the end of the file or a reserved word and the INvariable command does not occur, then all equations are assumed selected.
- The keyword INvariable is allowed. It should be followed by variable selection statements. In turn, DISPLAYCR selects all equations which have nonzero entries under the INvariable selections.
- The keyword INequation may be used. It should be followed by equation selection statements. In turn, DISPLAYCR selects all variables which have nonzero entries in the INequation selections.
- The keyword INTERSECT causes only coefficients at the intersection of the specified equations and variables to be displayed. This occurs for all specifications in this run of DISPLAYCR. One should use DISPLAYCR again if some intersecting and some non-intersecting displays are desired.
- When INTERSECT appears along with INvariable, the named variable is selected along with all the equations in which it falls. Similarly, when INTERSECT and INequation appear then all the named equations and the variables appearing in them are selected.

3.2 MATCHIT

Brief Purpose: MATCHIT retrieves the names and characteristics of selected variables and equations. The characteristics reported tell whether the items are nonlinear as well as reporting scaling characteristics and counts of the coefficients. MATCHIT will summarize the items which match a request or list all the items individually.

Usage Notes: The input to MATCHIT can include the keywords VARIABLE and EQUATION along with those keywords with the prefix LIST attached. When the LIST prefix is not used, the procedure summarizes the characteristics of all items which match the item requests counting the number of matching items, the number of those items which are nonlinear, the total coefficients under or in those items, the number of positive, negative, and nonlinear coefficients that fall under or in those items. This does not list the names of the individual items which match. If the LIST prefix is used (entering LISTVARIABLE
or LISTEQUATION) then the individual matching items are printed in the order in which they are encountered. For each matching item the information tells whether it is nonlinear, how many total coefficients it has, the count of positive, negative, and nonlinear coefficients falling under it, and the minimum and maximum absolute values of coefficients under it (excluding the objective function coefficient).

Note that when GAMS internal scaling features are employed then by default scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

**Input File:** This file contains the keyword MATCHIT, followed by optional item selection input data. The optional input identifies the variables and equations to be displayed. This input can contain the keywords VARIABLE or LISTVARIABLE followed by a specification of the variables to be selected using the procedure input specification conventions that were described above. This can be followed by the keyword EQUATION or LISTEQUATION and the specified entries.

Several special cases are relevant:

- If the procedure name is not followed by any selection input, then a count of all variables and equations appears.
- If the input is found, but the input does not begin with VARIABLE, EQUATION, LISTVARIABLE, or LISTEQUATION keywords, then the input is assumed to contain variable names.
- If the VARIABLE keyword is entered, but is not followed by variable selection statements, and LISTVARIABLE does not appear, then all variables are assumed selected.
- If the EQUATION or LISTEQUATION keyword cannot be found, then equations are assumed selected.
- If the EQUATION keyword is entered, but is not followed by equation selection statements or a LISTEQUATION entry, then all equations are assumed selected.
- The keyword LISTVARIABLE is allowed. It should be followed by variable selection statements. In turn, MATCHIT lists all variables which fall under the request.
- The keyword LISTEQUATION may also be used. It should be followed by equation selection statements. In turn, MATCHIT lists all equations which fall under the request.
- If none of the above keywords are found, the input is assumed to identify variables.

### 3.3 ANALYSIS

**Brief Purpose:** Analyzes the structure of all variables and equations. Information is given on errors involving obvious model misspecifications causing redundancy, zero variable values, infeasibility, unboundedness, or obvious constraint relaxations in linear programs. The checks are those identified in Tables 1, 2 and 3.

**Usage Notes:** The analysis tests given in Tables 1 and 2 are utilized to determine if individual variables or equations in the model possess obvious specification errors. One test, for example, considers whether or not in a maximization problem a variable appears which has a positive return in the objective function, but no coefficients in the constraints indicating an obviously unbounded model. Similarly, information is provided on whether certain equations can never be satisfied. For example, tests examine whether an equality equation appears with a negative right hand side and all positives on the left hand side. Also tests see whether the bounds on variables preclude equation satisfaction or make equations redundant (Table 3). In ANALYSIS these tests are applied to each and every variable and equation. The BLOCKPIC and BLOCKLIST routines utilize the tests on a block by block basis. Thus, the messages will be triggered only if every variable or equation in that block has the same problem. Also interactions between variables and equations are not checked so ANALYSIS only finds flaws contained in individual variables/equations.

**Input File:** The keyword ANALYSIS is all that is accepted.

### 3.4 BLOCKLIST

**Brief Purpose:** The BLOCKLIST procedure displays the number and characteristics of the items in each GAMS variable and equation block.

**Usage Notes:** The characteristic information gives:
1. The variable sign restriction or equation inequality type.
2. The number of variables or equations in this block;
3. The number of variables or equations with at least one nonlinear term in this block.
4. The number of positive coefficients under the variables or in the equations.
5. The number of negative coefficients under the variables or in the equations.
6. The number of nonlinear coefficients under the variables or in the equations.
7. The largest coefficient in absolute value in this block;
8. The smallest coefficient in absolute value in this block. Analysis tests are also performed as discussed under the ANALYSIS procedure.

Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

**Input File:** No input other than the procedure name is needed.

### 3.5 BLOCKPIC

**Brief Purpose:** Generates model schematics and scaling information. The schematics depict coefficient signs, total and average number of coefficients within each GAMS equation and variable block.

**Usage Notes:** These schematics are designed to aid users in identifying flaws in coefficient placement and sign. The summary information on problem scaling characteristics is designed to help users in scaling data. The scaling information is usually reported after any GAMS scaling (using the variablename.scale and equationname.scale features) but before solver scaling. (The user can change whether descaling is done - see the options file). Analysis tests are done using the procedures in Tables 1 and 2.

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

**Input File:** The keyword BLOCKPIC is all that is recognized.

### 3.6 PICTURE

**Brief Purpose:** Generates a schematic depicting the location, sign and magnitude of coefficients for selected variables and equations. Users can use this schematic to help identify flaws in coefficient placement, magnitude, or sign. Reports are also generated on the number of individual elements in the pictured portions of each variable and equation.

**Usage Notes:** This output can be quite large, so PICTURE should only be used for small models or model components. Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

**Input File:** Optional input instructions may appear after the PICTURE keyword. This input selects the variables and equations to be included. Only coefficients at the intersection of the selected variables and equations are portrayed. The selected item in the .GCK file can contain the keywords VARIABLE, or INVARIABLE followed by a specification of the selected variables using the procedure input specification conventions above. This can be followed by the keywords EQUATION or INEQUATION and the specified entries. Several special cases are also relevant:

- If the VARIABLE or INEQUATION keywords cannot be found, then all variables are assumed selected.
- If the EQUATION or INVARIABLE keywords cannot be found, then all equations are assumed to selected.
- If the none of the VARIABLE, INVARIABLE, EQUATION, or INEQUATION keywords are found, everything is pictured and all other input is ignored.
• When the INvariable keyword is used, then all equations in which those variables have coefficients are selected along with the named variables.

• When the INEQUATION keyword is used, then all variables which have coefficients in the named equations are selected along with the named equations.

3.7 POSTOPT

Brief Purpose: Does post optimality computations. In that capacity POSTOPT either:

• Reconstructs the reduced cost of variables after a GAMS model solution. Modelers can use this information to discover why certain variables are nonbasic or why certain shadow prices take on particular values, or

• Reconstructs the usage and supply across an equation after a GAMS model solution. Modelers can use this information to discover why certain variables or slacks take on particular values, as well as to find out where items within equations are produced and/or used.

Usage Notes: POSTOPT uses essentially the same input conventions as does DISPLAYCR. Thus, the usage notes in that selection are also relevant here. In addition:

1. POSTOPT requires a solution has been obtained GAMSCHK will automatically cause a solver to be invoked unless suppressed by the options file;

2. Nonlinear terms may not be accurate in the row sums as their marginal value not their total value is used but GAMS will have adjusted the right-hand sides for their presence; and

3. Attention can be restricted to only certain types of variables or equations. Variables that are INSOLUTION (Nonzero or with Zero marginals), NOTINSOLUTION (zero with a nonzero marginal) can be requested, BINDING or NON-BINDING equations can be focused on.

Note that when GAMS internal scaling features are employed, the default option is that the unscaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: An optional input file is read in, indicating the specific variables desired using the conventions explained under DISPLAYCR above. In addition:

• One can enter INSOLUTION to restrict attention to variables which are nonzero or have zero marginals.

• One can enter NOTINSOLUTION to restrict attention to zero variables.

• The above entries restrict alteration in all VARIABLE or INEQUATION selection statements in a POSTOPT run.

• One can enter BINDING to only consider equations with zero slack. Similarly, NONBINDING considers equations with nonzero slack.

• The above equation specifications restrict all sections by all EQUATION or INVARIABLES items in a POSTOPT run.

3.8 ADVISORY

Brief Purpose: To identify variables which could be unbounded or equations and variable bounds which could cause a model to be infeasible.

Usage Notes: The ADVISORY procedure causes a presolution report on the set of all: a) variables which could be unbounded and/or b) equations and variable bounds which could cause infeasibility. The tests used are summarized in Table 3. This procedure identifies all variables which would need to be bounded as well as all constraints which need artificial variables if one wishes to diagnose problems in a model. The same output is also generated by NONOPT but the ADVISORY version does not require a solution.

Input file: Just the word ADVISORY
3.9 NONOPT

**Brief Purpose:** To help diagnose unbounded and infeasible models.

**Usage Notes:** The NONOPT procedure can be used in either an informative mode or with models which terminate as unbounded or infeasible. NONOPT will look through an optimal model reporting all variables which may be potentially unbounded or infeasible and all equations which may be infeasible using the checks explained under the ADVISORY section. Also in an unbounded model NONOPT can report the names of unbounded or infeasible variables or equations as well as either budgeting or row summing them. NONOPT runs after a solution and causes a solve to occur.

**Input File:** NONOPT may be followed by optional keywords IDENTIFY or VERBOSE. The IDENTIFY keyword causes GAMSCHK to report potential unbounded variables and/or infeasible equations. VERBOSE causes full budgets and row summing as done by the POSTOPT procedure on infeasible equations, and/or variables as well as unbounded variables and/or equations. Only the last encountered of the VERBOSE or IDENTIFY keywords will be obeyed. The details on these options are as follows:

1. If the IDENTIFY keyword is used, then the rules in Table 3 are applied to the model. Identify also anticipates that large upper bounds and/or artificial variables may be present. In an optimal condition all variable and equation levels that have exponents greater than the user supplied level filter in the options file (or 6 by default) are identified as items which could be involved with an unbounded model. Similarly, all variables or equations with marginals greater in exponent than the user supplied marginal exponent filter will be identified as items potentially involved with an infeasible model.

2. When the VERBOSE keyword is read then all variables and equations which are listed as nonoptimal or infeasible are treated using the budgeting and row summing aspects of POSTOPT.

3. When no keyword is found and the model solution is not optimal then the nonoptimal equations, infeasible equations and/or nonoptimal variables automatically listed.

4 Options File

GAMSCHK accepts an option file controlling solver choice (when needed); descaling; and size of the nonoptimal filters; the number of variable and column blocks selection entries allowed. The file is called GAMSCHK.OPT

4.1 Solver Choice Options

GAMSCHK calls for the solution of the problem when the POSTOPT or NONOPT procedures are used. In doing this, GAMSCHK internally selects the default GAMS solver for a problem class. Users may override this choice using the solver options file. Users may also force or suppress the solution process.

There are 9 solver related keywords allowed in the options file. These are as follows:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>Gives name of solver for LP problems</td>
</tr>
<tr>
<td>NLP</td>
<td>Gives name of solver for NLP problems</td>
</tr>
<tr>
<td>MIP</td>
<td>Gives name of solver for MIP problems</td>
</tr>
<tr>
<td>DNLP</td>
<td>Gives name of solver for DNLP problems</td>
</tr>
<tr>
<td>SOLVERNAME</td>
<td>Gives name of solver to be used regardless of problem type</td>
</tr>
<tr>
<td>NOSOLVE</td>
<td>Suppresses solution of the problem</td>
</tr>
<tr>
<td>SOLVE</td>
<td>Forces solution of the problem</td>
</tr>
<tr>
<td>DESCALE</td>
<td>Controls treatment of scaling</td>
</tr>
<tr>
<td>OPTFILE</td>
<td>Solver options file number</td>
</tr>
</tbody>
</table>
In the first five cases, the option name is followed by the name of one of the licensed solvers. If the options file is empty, then the default solver will be used. If a solver name is given, then that solver will be used provided it matches the name of a solver GAMS recognizes.

4.2 When Should I Use SOLVE or NOSOLVE

Ordinarily GAMSCHK will cause a solver to be used if either the POSTOPT or the NONOPT options are used. However, users can force solutions under other cases or suppress solutions if desired.

One should only force a solution (using the SOLVE option) when one wishes to use the solution information after GAMSCHK is done either to examine the solution output or do post optimality calculations. Forcing a solution will not cause GAMSCHK to have improved representations of nonlinear terms. That will only occur when a SOLVE statement is executed before the SOLVE statement involving GAMSCHK.

4.3 Control of Number of Variable and Row Selections Allowed

The GAMSCHK program uses an upper estimate on the number of variable or equation blocks. In rare circumstances users may wish to override this choice. The options for this are:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARBLOCK</td>
<td>Maximum number of variable blocks allowed</td>
</tr>
<tr>
<td>EQUBLOCK</td>
<td>Maximum number of equation blocks allowed</td>
</tr>
</tbody>
</table>

These options are followed by a number, but should not be routinely used.

4.4 Scaling

GAMS users may be utilizing internal features which involve scaling through the Modelname.SCALEOPT=1, Variable-Name.SCALE, and EquationName.SCALE options. GAMSCHK can work with these options to create output which reflects scaled, unscaled or partially unscaled output. In particular, the command DESCALE can be entered with one of three options: NEVER, ALL, or PART. If you enter NEVER, then none of the model output will be descaled. If you enter ALL, then all of the model output will be descaled. The third option is to use PART. In that case the NONOPT and POSTOPT output will be descaled whereas scaled information will be displayed for PICTURE, BLOCKPIC, BLOCKLIST, MATCHIT and DISPLAYCR. The PART option allows investigation of scaling. If you do not enter a DESCALE option then all information will be reported as if the PART option was chosen.

4.5 NONOPT Filters

The NONOPT model in "IDENTIFY" mode checks through a model solution to identify large marginals and/or large variable values. The limits on these checks are provided by two options:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVELFILT</td>
<td>Numerical value of exponent on &quot;unbounded levels&quot;</td>
</tr>
<tr>
<td>MARGFILT</td>
<td>Numerical value of exponent on &quot;infeasible marginals&quot;</td>
</tr>
</tbody>
</table>

These options provide upper bounds on the exponents of the absolute values for the levels and marginals. They are followed by an integer which gives the exponent. Thus, entries like

```
LEVELFILT  7
MARGFILT   7
```

will cause the reporting of all marginals and levels which are greater in absolute value than $10^7$. 
4.6 Example Options File

The GAMSCHK option file is called GAMSCHK.OPT. An example of a file could look like the following 6 lines:

```
LP OSL
MIP LAMPS
VARBLOCK 50
SOLVE
DESCALE PART
LEVELFILT 4
```

4.7 Solver Options File

One other important aspect regarding the options file involves the use of a problem solver options file when a solver such as MINOS5, OSL, LAMPS etc. is also being used. As seen above the GAMSCHK.OPT does not recognize option commands such as those which would be submitted to the programming model solvers - OSL for example. In all cases GAMSCHK will cause the default option file for the solver to be used when invoking the solver. Thus if MINOS5 and the options file is invoked is being used, MINOS5 options are controlled by the option file MINO5.OPT while GAMSCHK.OPT controls GAMSCHK operation. Users can change the number of the solver options file being used by using the OPTFILE parameter in the options file. OPTFILE 2 would cause use of solver options file .OP2.

5 Known Bugs

There are a few bugs that can cause GAMSCHK to report improper outputs or results. A list of the known bugs, their symptoms and a remedy is given below.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Cause</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero Shadow Prices in POSTOPT</td>
<td>Old GAMS version or no Prior Solve</td>
<td>1) Make sure the model was solved, 2) if it was, do not suppress solve in option file, or 3) update to most recent GAMS version</td>
</tr>
<tr>
<td>Descaling Does Not Work</td>
<td>Old Version of GAMS</td>
<td>Update</td>
</tr>
<tr>
<td>GAMS Blows up after GAMSCHK Runs</td>
<td>Old GAMS version</td>
<td>Ignore, *.LST file, results are fine, can be fixed by updating to the most recent version of GAMS</td>
</tr>
<tr>
<td>POSTOPT has error in budgets equal to twice objective function coefficient for nonlinear maximizations</td>
<td>Old GAMS MINOS version</td>
<td>Switch to a minimization formulation or update GAMS/MINOS</td>
</tr>
<tr>
<td>ROWSUM does not fully account for the value of nonlinear terms in POSTOPT</td>
<td>Value of nonlinear terms sent from GAMS are only a marginal value</td>
<td>None planned. GAMSCHK would need reprogramming</td>
</tr>
<tr>
<td>Error message about size of VAR-BLOCK or EQNBLOCK</td>
<td>exceeded maximum number of blocks</td>
<td>Modify option file, enlarging or eliminating parameters</td>
</tr>
<tr>
<td>GAMSCHK won’t run</td>
<td>Files are not properly installed</td>
<td>Recheck installation. If still doesn’t work report to author</td>
</tr>
<tr>
<td>Zero shadow prices when using NO-SOLVE</td>
<td>Old version of GAMS solvers or Shadow prices suppressed</td>
<td>Try changing GAMSCOMP.TXT lines 2 or 0 to 12 or 10, if that doesn’t work update GAMS.</td>
</tr>
</tbody>
</table>
# Tables

## Table 1: Conditions under which a modeler should be advised of potential difficulty for equations without nonlinear terms.

<table>
<thead>
<tr>
<th>Type of constraint</th>
<th>Count of coefficients under a variable of this type with a particular sign</th>
<th>Sign of RHS</th>
<th>Type of PS /</th>
<th>Examples b/</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nonnegative</td>
<td>Nonpositive</td>
<td>Unrestricted</td>
<td></td>
</tr>
<tr>
<td>≤</td>
<td>≥0 a/</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>≥0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>≥0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>=</td>
<td>≥0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>≥0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>≥0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>≥0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>≥</td>
<td>0</td>
<td>≥0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>≥0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>≥0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **a/** The PS cases indicate, because the variables in this equation follow this pattern, that:
  1. The variables appearing with nonzeros in this equation are forced to equal zero.
  2. This equation can never be satisfied and is obviously infeasible.
  3. This equation is redundant. The nonnegativity conditions are a stronger restriction.
- **b/** In the examples x denotes indexed non-negative variables, y indexed non-positive variables, and z a single unrestricted variable.
- **c/** Here and in the cases below at least one nonzero must occur.
- **d/** These entries give examples of the problem covered by each warning. Namely, in the first case examining only the nonnegative variables suppose all those variables have signs ≥ 0 but the right-hand-side is zero. Thus, we have \( X \geq 0 \) and \( X \leq 0 \) which implies \( X = 0 \). A warning is generated in that case.
- **e/** Only one coefficient is allowed.
Table 2: Conditions under which a modeler should be warned about variables in a maximization problem.

<table>
<thead>
<tr>
<th>Type of Variable</th>
<th>Objective function coefficient sign</th>
<th>Number of $a_{ij}$'s of a sign in $\geq$ rows $=$ rows $\leq$ rows</th>
<th>PS$^a/ b/ c/</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>+      -      +      -      +      -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonnegative</td>
<td></td>
<td>$\geq 0$</td>
<td>0     0        0        0         $\geq 0$</td>
<td>Unbounded Variable case 1</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0     $\geq 0$</td>
<td>0        0        $\geq 0$</td>
<td>Zero optimal solution case 2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$\geq 0$</td>
<td>0     0        0        0         $\geq 0$</td>
<td>Variable Relaxes constraint case 3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$\geq 0$</td>
<td>0     $\geq 0^/$</td>
<td>$\geq 0^/$</td>
</tr>
<tr>
<td>Nonpositive</td>
<td>-</td>
<td>0     $\geq 0$</td>
<td>0        0        $\geq 0$</td>
<td>Unbounded Variable case 1</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>$\geq 0$</td>
<td>0     0        0        0         $\geq 0$</td>
<td>Zero optimal solution case 2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0     $\geq 0$</td>
<td>0        0        $\geq 0$</td>
<td>Variable Relaxes constraint case 3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>$\geq 0$</td>
<td>0     $\geq 0^/$</td>
<td>$\geq 0^/$</td>
</tr>
<tr>
<td>Unrestricted</td>
<td>+/-</td>
<td>0     0     0        0        0         0         0</td>
<td>Unbounded Variable case 1</td>
<td>max $\pm z$</td>
</tr>
</tbody>
</table>

- $a/ PS$ cases are: The variables which satisfy this condition are:
  1. Unbounded as they contribute to the objective function while satisfying the constraints.
  2. Obviously zero since they consume constraint resources and have a cost in the objective function.
  3. Warning this variable relaxes all constraints in which it appears
  4. Warning this variable relaxes all the equality constraints in which it appears in one direction

- $b/ Here x(y) has a positive objective term and can be increased without ever violating any constraints so x(y) is unbounded.

- $c/ Only one coefficient can be present in the equality rows
Table 3: Conditions When Model Elements Could be Unbounded or Infeasible.

**Conditions for Potential Unbounded Variables – Presence of Bounds**

<table>
<thead>
<tr>
<th>Types of Variables</th>
<th>Sign of Objective in Max Problem</th>
<th>Upper</th>
<th>Lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0 /</td>
<td>+</td>
<td>None</td>
<td>⋄</td>
</tr>
<tr>
<td>≤ 0</td>
<td>-</td>
<td>⋄</td>
<td>⋄</td>
</tr>
<tr>
<td>unrestricted</td>
<td>+</td>
<td>None</td>
<td>⋄</td>
</tr>
<tr>
<td>unrestricted</td>
<td>⋄</td>
<td>⋄</td>
<td>None</td>
</tr>
</tbody>
</table>

- **a/** If a non negative variable has a positive objective function coefficient without an upper bound, then the variable could be unbounded.
- **b/** Any reasonable value can exist for this item

**Conditions for Potential Infeasibility Caused by Bounds on Variables**

<table>
<thead>
<tr>
<th>Types of Variables</th>
<th>Existence of Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥c/0</td>
<td>+</td>
</tr>
<tr>
<td>≤ 0</td>
<td>⋄</td>
</tr>
<tr>
<td>unrestricted</td>
<td>+</td>
</tr>
<tr>
<td>unrestricted</td>
<td>⋄</td>
</tr>
</tbody>
</table>

- **c/** If a nonnegative variable has a positive lower bound then it could cause infeasibility.

**Conditions for Potential Infeasibility in Equations**

<table>
<thead>
<tr>
<th>Type of Equations</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤d/</td>
<td>-</td>
</tr>
<tr>
<td>≥</td>
<td>+</td>
</tr>
<tr>
<td>=</td>
<td>+ or -</td>
</tr>
</tbody>
</table>

- **d/** When a less than or equal equation is present it may not be able to be satisfied if it has a negative RHS.
Table 4: Conditions for Potential Infeasibility or Redundancy in Equations Based on Bounds on Variables.

<table>
<thead>
<tr>
<th>TYPE OF CONSTRAINT</th>
<th>PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤b</td>
<td></td>
</tr>
<tr>
<td>≥b</td>
<td></td>
</tr>
<tr>
<td>&gt;b</td>
<td>INFEASIBLE</td>
</tr>
<tr>
<td>—</td>
<td></td>
</tr>
<tr>
<td>&gt;b</td>
<td>REDUNDANT</td>
</tr>
<tr>
<td>&lt;b</td>
<td>INFEASIBLE</td>
</tr>
<tr>
<td>—</td>
<td></td>
</tr>
<tr>
<td>&lt;b</td>
<td>REDUNDANT</td>
</tr>
</tbody>
</table>

Note:

- **a/** Suppose $X_j$ is bounded with $LB_j$ (lower bound) ≤ $X_j$ ≤ $UB_j$ (upper bound), and we have the sum evaluated at the lower bounds will be the smallest value which could happen in that sum. If the constraint is $< b$, then if the sum is $> b$, we know that this constraint will never be satisfied. If the constraint is $> b$, and the sum is $> b$, we know that this constraint will not limit any possible $X$ value. Hence, it is redundant.

- **b/** Suppose $X_j$ is bounded as follows, $LB_j$ (lower bound) ≤ $X_j$ ≤ $UB_j$ (upper bound), and we have the sum evaluated at the upper bounds which is either $> b$ or $< b$, in that sum. If the sum is $< b$, and the constraint holds it $< b$ then we know that this constraint will not limit any possible $X$ value. Hence, it is redundant. If the constraint holds it greater than $b$, but the sum is $< b$, we know that this constraint will never be satisfied.

- **c/** Thanks to Paul Preckel for bringing these tests to the authors’ attention.
Appendix A: Reserved Names

VARIABLE
VARIABLES
EQUATION
EQUATIONS
INVARIABLE
INVARIABLES
INEQUATION
INEQUATIONS
LISTVARIABLE
LISTVARIABLES
LISTEQUATION
LISTEQUATIONS
POSTOPT
DISPLAYCR
PICTURE
BLOCKPIC
ANALYSIS
MATCHIT
BLOCKLIST
NONOPT
INSOLUTION
NOTINSOLUTION
NONINSOLUTION
VERBOSE
ADVISORY
BINDING
NONBINDING
INTERSECT
IDENTIFY
Pw=
## Appendix B: GAMSCHK One Page Summary

### Invoking GAMSCHK

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Allowed SubKEYWORDS</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAYCR</td>
<td>VARIABLE*</td>
<td>Displays coefficients of selected variables and equations</td>
</tr>
<tr>
<td></td>
<td>INVARIABLE*</td>
<td>Indicates variable selections follow</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Indicates equations are wanted in which selected variables fall</td>
</tr>
<tr>
<td></td>
<td>INEQUALATION*</td>
<td>Indicates equation selections follow</td>
</tr>
<tr>
<td></td>
<td>INTERSECT++</td>
<td>Shows coefficients which appear at intersections of selected var/eqn</td>
</tr>
<tr>
<td>MATCHIT</td>
<td>VARIABLE*</td>
<td>List variable and equation names and summarize characteristics</td>
</tr>
<tr>
<td></td>
<td>LISTVariable*</td>
<td>Summarizes all variables matching selection statements</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Lists each variable matching a selection statement</td>
</tr>
<tr>
<td></td>
<td>LISTEQUATION*</td>
<td>Lists each equation matching a selection statement</td>
</tr>
<tr>
<td>ANALYSIS</td>
<td></td>
<td>Checks for obvious structural defects</td>
</tr>
<tr>
<td>BLOCKLIST</td>
<td></td>
<td>Summarizes characteristics of variable and equation blocks</td>
</tr>
<tr>
<td>BLOCKPIC</td>
<td></td>
<td>Generates block level schematics</td>
</tr>
<tr>
<td>PICTURE</td>
<td>VARIABLE*</td>
<td>Generates tableau schematics</td>
</tr>
<tr>
<td></td>
<td>INVARIABLE*</td>
<td>Indicates variable selections follow</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Indicates equation selections follow</td>
</tr>
<tr>
<td></td>
<td>INEQUALATION*</td>
<td>Indicates variables are wanted that fall in selected equations</td>
</tr>
<tr>
<td>POSTOPT</td>
<td>VARIABLE*</td>
<td>Reconstructions reduced cost and equation activity</td>
</tr>
<tr>
<td></td>
<td>INVARIABLE*</td>
<td>Indicates variable selections follow</td>
</tr>
<tr>
<td></td>
<td>EQUATION*</td>
<td>Indicates equation selections follow</td>
</tr>
<tr>
<td></td>
<td>INEQUALATION*</td>
<td>Indicates variables are wanted that fall in selected equations</td>
</tr>
<tr>
<td></td>
<td>INTERSECT++</td>
<td>Shows coefficients which appear at intersections of selected var/eqn</td>
</tr>
<tr>
<td></td>
<td>NOTINSOLUTION++</td>
<td>Only nonzero vars or those with zero reduced cost</td>
</tr>
<tr>
<td></td>
<td>INSOLUTION++</td>
<td>Only zero vars will be selected</td>
</tr>
<tr>
<td></td>
<td>BINDING++</td>
<td>Only eqns with zero slack will be computed</td>
</tr>
<tr>
<td></td>
<td>NONBINDING++</td>
<td>Only eqns with nonzero slack will be computed</td>
</tr>
<tr>
<td>ADVISORY</td>
<td></td>
<td>List potential infeasible and unbounded items</td>
</tr>
<tr>
<td>NONOPT</td>
<td>IDENTIFY</td>
<td>Lists potential or actual nonoptimal items</td>
</tr>
<tr>
<td></td>
<td>VERBOISE</td>
<td>Same as ADVISORY but after solution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Does POSTOPT computations on nonoptimals</td>
</tr>
</tbody>
</table>

### Other Notes

- Items marked above with an * are followed by item selection statements.
- Items marked with ++ modify the types of variables, equations and coefficients selected.
- In item selection an * is a wild card for multiple characters while a . is a wildcard for one character.
- Spaces and capitalization don’t matter in any of the input.
- Options file controls scaling, solver choice, nonopt filters and maximum allowed selections.
- Page width is controlled by a PW= keyword but cannot exceed GAMS page width.
- Lines beginning with a ? or a # are treated as comments.
9 Appendix C: Summary of GAMSCHK Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNS</td>
<td>solver for CNS problems</td>
<td></td>
</tr>
<tr>
<td>DESCAL</td>
<td>controls treatment of scaling</td>
<td>part</td>
</tr>
<tr>
<td>DNLP</td>
<td>solver for DNLP problems</td>
<td></td>
</tr>
<tr>
<td>EQUBLOCK</td>
<td>maximum number of equation blocks allowed</td>
<td>-5</td>
</tr>
<tr>
<td>LEVELFILT</td>
<td>numerical value of exponent on &quot;unbounded levels&quot;</td>
<td>6</td>
</tr>
<tr>
<td>LP</td>
<td>solver for LP problems</td>
<td></td>
</tr>
<tr>
<td>MARGFILT</td>
<td>numerical value of exponent on &quot;infeasible marginals&quot;</td>
<td>6</td>
</tr>
<tr>
<td>MCP</td>
<td>solver for MCP problems</td>
<td></td>
</tr>
<tr>
<td>MINLP</td>
<td>solver for MINLP problems</td>
<td></td>
</tr>
<tr>
<td>MIP</td>
<td>solver for MIP problems</td>
<td></td>
</tr>
<tr>
<td>MPEC</td>
<td>solver for MPEC problems</td>
<td></td>
</tr>
<tr>
<td>NLP</td>
<td>solver for NLP problems</td>
<td></td>
</tr>
<tr>
<td>NOSOLVE</td>
<td>suppresses solution of the problem</td>
<td></td>
</tr>
<tr>
<td>OPTFILE</td>
<td>solver options file number</td>
<td></td>
</tr>
<tr>
<td>RMINLP</td>
<td>solver for RMINLP problems</td>
<td></td>
</tr>
<tr>
<td>RMIP</td>
<td>solver for RMIP problems</td>
<td></td>
</tr>
<tr>
<td>RMPEC</td>
<td>solver for RMPEC problems</td>
<td></td>
</tr>
<tr>
<td>SOLVE</td>
<td>forces solution of the problem</td>
<td></td>
</tr>
<tr>
<td>SOLVERNAME</td>
<td>solver for any problems</td>
<td></td>
</tr>
<tr>
<td>VARBLOCK</td>
<td>maximum number of variable blocks allowed</td>
<td>-5</td>
</tr>
</tbody>
</table>

10 GAMSCHK References

GAMS/KESTREL - Remote Solver Execution on NEOS Servers

Contents

1 Background .................................................. 777
2 Using GAMS/KESTREL ..................................... 777

1 Background

The Kestrel client/server is a way of sending your GAMS solve job to be solved via the NEOS Server from within your usual GAMS modeling environment and receiving results that can be processed as with any local solver. Starting with distribution 23.7 the Kestrel solver is part of the GAMS distribution. For older versions of GAMS a download can be found at the Kestrel page on the NEOS Server. The solve statement using the GAMS/KESTREL solver invokes a client program that sends your problem to a solver running on one of the NEOS Server’s remote computers. The results from the NEOS Server are eventually returned through Kestrel to GAMS, where you can view and manipulate them locally in the usual way. The introduction below covers everything you need to know to start using GAMS/KESTREL. Further information about more advanced features and other uses of Kestrel can be found at the Kestrel page on the NEOS Server.

2 Using GAMS/KESTREL

The Kestrel solver can be used to solve a GAMS model remotely. For example, consider the transport model from GAMS Model Library. It can be solved locally in GAMS through the following statements,

model transport /all/;
solve transport using lp minimizing z;

which specify the transport model and solve it with the default linear programming solver. We can add an option statement to the code to explicitly specify the solver. For example, if we change the linear programming solver to GAMS/MINOS, the code becomes

model transport /all/;
option lp=minos;
solve transport using lp minimizing z;

To solve the same problem remotely through the NEOS Server, we simply change the linear programming solver to Kestrel.

transport.optfile=1;
option lp=kestrel;
solve transport using lp minimizing z;
$echo kestrel_solver minos > kestrel.opt
The statement `transport.optfile=1` specifies that an options file, called `kestrel.opt`, will be used. The options file contains the remote solver name as well as any options for the remote solver. We instruct the Kestrel solver to use GAMS/MINOS as the remote solver by writing the following `kestrel.opt` file:

```
kestrel_solver minos
```

If you do not know what solvers are available via GAMS/KESTREL, submitting a job with a nonexistent solver set will return a list of enabled solvers.

A subsequent run of the code through the GAMS interpreter results in the transport model being solved through the NEOS Server with the GAMS/MINOS solver. Once the job is submitted to the NEOS Server, a job number, password, and Web address are displayed to the screen, which provide information on accessing the job and viewing the intermediate output, for example,

```
--- Executing KESTREL
Job has been submitted to Kestrel
Kestrel/NEOS Job number    : 93478
Kestrel/NEOS Job password : utiwtxTK
Check the following URL for progress report :
  http://www-neos.mcs.anl.gov/neos/neos-cgi/check-status.cgi?job=93478&pass=utiwtxTK
  In case of problems, e-mail: neos-comments@mcs.anl.gov
```

To look at the solver’s output while it is running, point your browser at the URL given in the Kestrel output as shown above, and click on View Intermediate Results in the web page that appears. This will take you to another page that shows all of the output produced by the solver for your problem so far. To track the solver’s progress, simply update this page periodically.

If the NEOS Server or the network becomes unavailable after the submission, a particular job can be retrieved by setting both the kestrel job and kestrel password in the options file.

```
kestrel_solver minos
kestrel_job 93478
kestrel_password utiwtxTK
```

Re-issuing the command `gams transport` with this options file will retrieve the results for the specified job number.
GAMS/LINGO

Contents

<table>
<thead>
<tr>
<th>1</th>
<th>Introduction</th>
<th>779</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Lingo Path</td>
<td>779</td>
</tr>
<tr>
<td>3</td>
<td>GAMS/LINGO Options</td>
<td>780</td>
</tr>
<tr>
<td>3.1</td>
<td>General options</td>
<td>780</td>
</tr>
<tr>
<td>3.2</td>
<td>Other options</td>
<td>780</td>
</tr>
</tbody>
</table>

1 Introduction

GAMS/LINGO allows users to solve GAMS models using solvers within the LINDO modeling system. The GAMS/LINGO link comes free with any GAMS system. Users must have a licensed LINGO system installed and have the LINGO executable in their path.

To run GAMS/LINGO, just specify the solver as lingo. For example, if we wish to solve the transport.gms model, we would run

```
>> gams transport.gms lp=lingo;
```

As for other GAMS solvers, options can be passed on via solver option files. GAMS/LINGO specific options are described in the section GAMS/LINGO Options.

By default, GAMS/LINGO returns a model status of 14 (no solution) and a solver return status of 1 (normal completion), provided the link is executed normally. This includes the case where the LINGO executable is not found.

2 Lingo Path

GAMS searches for a LINGO executable using the following hierarchy:

- Via the options LingoPath and RunLingo within a GAMS/LINGO solver option file.
- An lingopath.txt file located in the GAMS system directory specifying the path of the LINGO executable.
- The system path.

For example, GAMS will first search for the LINGO executable within the lingo.opt file, if specified. If not found, it will search within the GAMS system directory for a file called lingopath.txt specifying the LINGO directory. Finally if lingopath.txt is not found, the GAMS will try the system path.

If no LINGO executable is found, the user will see a message similar to

LINGO Link 0. Jan 26, 2005 LNX.00.NA 21.6 002.000.000.LXI P3PC
--- No LingoPath option or "lingopath.txt" file found
--- System PATH will be used

sh: line 1: runlingo: command not found

The last line is platform dependent but indicates that LINGO was not found, either because it is not installed or because it is not found in the system path.

3 GAMS/LINGO Options

GAMS/LINGO solver options are passed on through solver option files. If you specify

<modelname>.optfile = 1;

before the SOLVE statement in your GAMS model, GAMS/LINGO will then look for and read an option file with the name lingo.opt (see "Using Solver Specific Options" for general use of solver option files).

3.1 General options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>IterLim</td>
<td>Minor iteration limit</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>LingoPath</td>
<td>Path to LINGO system files</td>
<td></td>
</tr>
<tr>
<td>OptCR</td>
<td>Relative termination for MIPs and Global Optimization Problems</td>
<td>GAMS OptCR</td>
</tr>
<tr>
<td>ResLim</td>
<td>Resource limit</td>
<td>GAMS ResLim</td>
</tr>
</tbody>
</table>

3.2 Other options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>DotLng</td>
<td>LINGO input file name</td>
<td>lingo.lng</td>
</tr>
</tbody>
</table>
The purpose of this chapter is to detail an extension of the GAMS modeling system that allows collections of models (parameterized exogenously by a set of samples or indices) to be described, instantiated, and solved efficiently.

As a specific example, we consider the parametric optimization problem \( \mathcal{P}(s) \) defined by:

\[
\min_{x \in X(s)} f(x; s) \quad \text{s.t.} \quad g(x; s) \leq 0 \tag{38.1}
\]

where \( s \in S = \{1, \ldots, K\} \). Note that each scenario \( s \) represents a different problem for which the optimization variable is \( x \). The form of the constraint set above is simply for concreteness; equality constraints and range and bound constraints are trivial extensions of the above framework. Clearly the problems \( \mathcal{P}(s) \) are interlinked. We intend to show how such problems can be easily specified within GAMS, and detail one type of algorithmic extension that can exploit the nature of the linkage. Other extensions of GAMS allow solves to be executed in parallel or by using grid computing resources. Note that in our description we will use the terms indexed, parameterized, and scenario somewhat interchangeably. An extended version of this chapter containing several examples is available as a paper at http://www.gams.com/modlib/adddocs/gusspaper.pdf.

2 Design Methodology

One of the most important functions of GAMS is to build a model instance from the collection of equations (i.e. an optimization model defined by the GAMS keyword MODEL) and corresponding data (consisting of the content of GAMS (sub)sets and parameters). Such a model instance is constructed or generated when the GAMS execution system executes a SOLVE statement. The generated model instance is passed to a solver which searches for a solution of this model instance and returns status information, statistics, and a (primal and dual) solution of the model instance. After the solver terminates, GAMS brings the solution back into the GAMS database, i.e. it updates the level (.L) and marginal (.M) fields of variable and equation symbols used in the model instance. Hence, the SOLVE statement can be interpreted as a complex operator against the GAMS database. The model instance generated by a SOLVE statement only lives during the execution of this one statement, and hence has no representation within the GAMS language. Moreover, its structure does fit the relational data model of GAMS. A model instance consists of vectors of bounds and right hand sides, a sparse matrix representation of the Jacobian, a representation of the non-linear expressions that allow the efficient calculation of gradient vectors and Hessian matrices, and so on.

This chapter is concerned with solving collections of models that have similar structure but modified data. As an example, consider a linear program of the form:

\[
\min \mathbf{c}^T \mathbf{x} \quad \text{s.t.} \quad A \mathbf{x} \geq b, \ell \leq \mathbf{x} \leq u.
\]
The data in this problem is \((A, b, c, \ell, u)\). Omitting some details, the following code could be used within GAMS to solve a collection of such linear programs in which each member of the collection has a different \(A\) matrix and lower bound \(\ell\):

```
1 Set i / ... /, j / ... /;
2 Parameter A(i,j), b(i);
3 Variable x(j), z, ...;
4 Equation e(i), ...;
5 e(i).. sum(j, A(i,j)*x(j)) =g= b(i);
6 ...
7 model mymodel /all/;
8
9 Set s / s1*s10 /;
10 Parameter A_s(s,i,j) Scenario data
11 xlo_s(s,j) Scenario lower bound for variable x
12 xl_s(s,j) Scenario solution for x.l
13 em_s(s,i) Scenario solution for e.m;
14 Loop(s,
15   A(i,j) = A_s(s,i,j);
16   x.lo(j)= xlo_s(s,j);
17   solve mymodel min z using lp;
18   xl_s(s,j) = x.l(j);
19   em_s(s,i) = e.m(i);
20);
```

Summarizing, we solve one particular model (mymodel) in a loop over \(s\) with an unchanged model rim (i.e. the same individual variables and equations) but with different model data and different bounds for the variables. The change in model data for a subsequent solve statement does not depend on the previous model solutions in the loop.

The purpose of this new Gather-Update-Solve-Scatter (GUSS) manager is to provide syntax at the GAMS modeling level that makes an instance of a problem that provides limited access to treat that instance as an object, and allows the modeler to update portions of it iteratively. Specifically, we provide syntax that gives a list of data changes to an instance, and allows these changes to be applied sequentially to the instance (which is then solved without returning to GAMS). Thus, we can simulate a limited set of actions to be applied to the model instance object and retrieve portions of the solution of these changed instances back in the modeling environment. Such changes can be done to any model type in GAMS, including nonlinear problems and mixed integer models. However, the only changes we allow are to named parameters appearing in the equations and lower and upper bounds used in the model definition.

Thus, in the above example GUSS allows us to replace lines 15-21 by

```
Set dict / s. scenario. ''
   A. param. A_s
   x. lower. xlo_s
   x. level. xl_s
   e. marginal. em_s /;
solve mymodel min z using lp scenario dict;
```

The three dimensional set dict (you can freely choose the name of this symbol) contains mapping information between symbols in the model (in the first position) and symbols that supply required update data or store solution information (in the third position), and the type of update/storing (in the second position). An exception to this rule is the tuple with label scenario in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. The remaining tuples in the set dict can be grouped into input and output tuples. Input tuples determine the modifications of the model instance prior to solving, while output tuples determine which part of the solution gets saved away. The following keywords can be used in the second position of the set dict:

<table>
<thead>
<tr>
<th>Type</th>
<th>Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>param</td>
<td>Supplies scenario data for a parameter used in the model</td>
</tr>
<tr>
<td></td>
<td>lower</td>
<td>Supplies scenario lower bounds for a variable</td>
</tr>
<tr>
<td></td>
<td>upper</td>
<td>Supplies scenario upper bounds for a variable</td>
</tr>
<tr>
<td></td>
<td>fixed</td>
<td>Supplies scenario fixed bounds for a variable</td>
</tr>
<tr>
<td>Output:</td>
<td>level</td>
<td>Stores the levels of a scenario solution of variable or equation</td>
</tr>
<tr>
<td></td>
<td>marginal</td>
<td>Stores the marginals of a scenario solution of variable or equation</td>
</tr>
</tbody>
</table>
Sets in the model cannot be updated. GUSS works as follows: GAMS generates the model instance for the original data. As with regular SOLVE statements, all the model data (e.g. parameter $a$) needs to be defined at this time. The model instance with the original data is also called the base case. The solution of the base case is reported back to GAMS in the regular way and is accessible via the regular .L and .M fields after the SOLVE statement. After solving the base case, the update data for the first scenario is applied to the model. The tuples with lower, upper, fixed update the bounds of the variables, whereas the tuples with param update the parameters in the model.

The scenario index $k$ needs to be the first index in the parameters mapped in the set dict. The update of the model parameters goes far beyond updating the coefficients of the constraint matrix/objective function or the right hand side of an equation, as one can do with some other systems. GAMS stores all the necessary expressions of the constraints with the model instance, so the change in the constraint matrix coefficient is the result of an expression evaluation. For example, consider a term in the calculation of the cost for shipping a variable amount of goods $x(i,j)$ between cities $i$ and $j$. The expression for shipping cost is $d(i,j) \times f \times x(i,j)$, i.e. the distance between the cities times a freight rate $f$ times the variable amount of goods. In order to find out the sensitivity of the solution with respect to the freight rate $f$, one can solve the same model with different values for $f$. In a matrix representation of the model one would need to calculate the coefficient of $x(i,j)$ which is $d(i,j) \times f$, but with GUSS it is sufficient to supply different values for $f$ that potentially result in many modified coefficients on the matrix level. GUSS evaluates the shipping cost term and communicates the resulting matrix coefficient to the solver reliably behind the scenes.

After the variable bound and the model parameter updates have been applied and the resulting updates to the model instance data structures (e.g. constraint matrix) has been determined, the modified model instance is passed to the solver. Some solvers (e.g. Cplex, Gurobi, and Xpress) allow modifying a model instance. In these cases GUSS only communicates the changes from the previous model instance to the solver. This reduces the amount of data communicated to the solver and also, in the case of an LP model, allows the solver to restart from an advanced basis and its factorization. In the case of an NLP model, this provides initial values. After the solver determines the solution of a model instance, GUSS stores the part of the solution requested by the output tuples of dict to some GAMS parameters and continues with the next scenario. GUSS emphasizes on speed and only works with solver that allow to communicate the model instance through memory. Hence, the following solvers cannot be used as subsolvers of GUSS: ALPHAECP, AMPL, BARON, BDMLP, BENCH, CONVERT, DECISC, DECISM, DICOPT, EXAMINER, GAMSCHK, JAMS, KESTREL, LINGO, LOGMIP, LS, MILES, MPECDUMP, MPSGE, MSNLP, NLPEC, OQNLP, PATHNLP, SBB, and XA.

3 GUSS Options

The execution of GUSS can be parameterized using some options. Options are not passed through a solver option file but via another tuple in the dict set. The keyword in the second position of this tuple is opt. A one dimensional parameter is expected in the first position (or the label ''). This parameter may contain some of the following labels with values:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptfileInit:</td>
<td>Option file number for the first solve (default from GAMS OptFile setting)</td>
</tr>
<tr>
<td>Optfile:</td>
<td>Option file number for subsequent solves (default 0)</td>
</tr>
<tr>
<td>LogOption:</td>
<td>Determines amount of log output:</td>
</tr>
<tr>
<td></td>
<td>0 - Moderate log (default)</td>
</tr>
<tr>
<td></td>
<td>1 - Minimal log</td>
</tr>
<tr>
<td></td>
<td>2 - Detailed log</td>
</tr>
<tr>
<td>NoHotStart:</td>
<td>Disable hot start capability in solver that supports hot starts (default 0)</td>
</tr>
<tr>
<td>NoMatchLimit:</td>
<td>Limit of unmatched scenario records (default 0)</td>
</tr>
<tr>
<td>RestartType:</td>
<td>Determines restart point for the scenarios</td>
</tr>
<tr>
<td></td>
<td>0 - Restart from last solution (default)</td>
</tr>
<tr>
<td></td>
<td>1 - Restart from solution of base case</td>
</tr>
<tr>
<td></td>
<td>2 - Restart from input point</td>
</tr>
<tr>
<td>SkipBaseCase:</td>
<td>Switch for solving the base case (0 solves the base case)</td>
</tr>
<tr>
<td>SolveEmpty:</td>
<td>Limit of solved empty scenarios, afterwards scenarios will be skipped (default 0)</td>
</tr>
</tbody>
</table>
Gather-Update-Solve-Scatter (GUSS)

**UpdateType:**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Scenario update mechanism:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Set everything to zero and apply changes (default)</td>
</tr>
<tr>
<td>1</td>
<td>Reestablish base case and apply changes</td>
</tr>
<tr>
<td>2</td>
<td>Build on top of last scenario and apply changes</td>
</tr>
</tbody>
</table>

For the example model above the `UpdateType` setting would mean:

- **UpdateType=0:** \( \text{loop}(s, \ A(i,j) = A_s(s,i,j)) \)
- **UpdateType=1:** \( \text{loop}(s, \ A(i,j) = A_{base}(i,j); \ A(i,j) \# A_s(s,i,j)) \)
- **UpdateType=2:** \( \text{loop}(s, \ A(i,j) \# A_s(s,i,j)) \)

The option `SkipBaseCase=1` allows the user to skip the base case. This means only the scenarios are solved and there is no solution reported back to GAMS in the traditional way. The third position in the opt-tuple can contain a parameter for storing the scenario solution attribute information, e.g. model and solve status, or needs to have the label `''`. The labels to store solution status information must be known to GAMS, so one needs to declare a set with such labels. A convenient way to enter these attributes is via `System.GUSSModelAttributes`:

```plaintext
Set ma GUSS Model Attributes / System.GUSSModelAttributes /; display ma;

---- 1 SET ma GUSS Model Attributes

ModelStat, SolveStat, NumInfes, NumNOpt, SumInfeS, IterUsd
ResUsd , ObjVal , NodUsd , ObjEst , DomUsd , RObj

The following example shows how to use some of the GUSS options and the use of a parameter to store some solution status information:

Set h solution headers / System.GUSSModelAttributes /;
Parameter
  o / SkipBaseCase 1, UpdateType 1, Optfile 1 /
  r_s(s,h) Solution status report;
Set dict / s. scenario. ''
  o. opt. r_s
  a. param. a_s
  x. lower. xl_s
  x. level. xl_s
  e. marginal. em_s /;
solve mymodel min z using lp scenario dict;
```

Please note that the domain set of the solution status report attributes (here \( h \)) must only contain model attributes known to GUSS. If this domain (unless the domain in `*`) contains a label unknown to GUSS, a compilation error is triggered.

### 4 Implementation Details

This section describes some technical details that may provide useful insight in case of unexpected behavior.

Because GUSS changes all model parameters mentioned in the `dict` set to variables, a linear model can produce some non-linear instructions (e.g. \( d(i,j) \times f \times x(i,j) \) becomes a non-linear expression since \( f \) becomes a variable in the model instance given to GUSS). This also explains why some models compile without complaint, but if the model is used in the context of GUSS, the compile time check of the model will fail because a parameter that is turned into a variable can no longer be used in that way. For example, suppose the model contains a constraint \( e(i) .. \sum(j \# A(i,j), \ldots) \). If \( A(i,j) \) is a parameter in the regular model, the compiler will not complain, but if \( A \) becomes a parameter that shows up
in the first position of a param tuple in the dict set, the GAMS compiler will turn A into a variable and complain that an endogenous variable cannot be used in a $-condition.

The sparsity pattern of a model can be greatly affected by GUSS. In a regular model instance GAMS will only generate and pass on non-zero matrix elements of a constraint e(i).. sum(j, A(i,j)*x(j)) ... so the sparsity of A determines the sparsity of the generated model instance. GUSS allows to use this constraint with different values for A hence GUSS cannot exclude any of the pairs (i,j) and generate a dense matrix. The user can enforce some sparsity by explicitly restricting the (i,j) pairs: e(i).. sum(ij(i,j), A(i,j)*x(j)) ...

The actual change of the GAMS language required for the implementation of GUSS is minimal. The only true change is the extension of the SOLVE statement with the term SCENARIO dict. Existing language elements have been used to store symbol mapping information, options, and model result statistics. Some parts of the GUSS presentation look somewhat unnatural, e.g. since dict is a three dimensional set the specification the scenario set using keyword scenario requires a third dummy label "". However, this approach gives maximum flexibility for future extension, allows reliable consistency checks at compile and execution time, and allows the user to delay the commitment for significant and permanent syntax changes of a developing method to handle model instances at a GAMS language level.
COIN-OR IPOPT (Interior Point Optimizer) is an open-source solver for large-scale nonlinear programming (NLP). The code has been written primarily by Andreas Wächter.

IPOPT implements an interior point line search filter method for nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. For more information on the algorithm we refer to [6, 9–12] and the IPOPT web site. Most of the IPOPT documentation in the section was taken from the IPOPT manual [5].

| Contents |
|------------------|---------------------|------------------|
| 1 | The linear solver in IPOPT | 788 |
| 2 | Usage | 788 |
| 2.1 | Using Harwell Subroutine Library routines with GAMS/IPOPT. | 788 |
| 2.2 | Specification of Options | 788 |
| 2.3 | Warmstarting Ipopt | 789 |
| 3 | Output | 789 |
| 3.1 | Diagnostic Tags for IPOPT | 792 |
| 4 | List of IPOPT Options | 793 |
| 4.1 | Barrier Parameter Update | 793 |
| 4.2 | Convergence | 794 |
| 4.3 | Hessian Approximation | 794 |
| 4.4 | Initialization | 795 |
| 4.5 | Line Search | 795 |
| 4.6 | Linear Solver | 796 |
| 4.7 | MA27 Linear Solver | 796 |
| 4.8 | MA28 Linear Solver | 797 |
| 4.9 | MA57 Linear Solver | 797 |
| 4.10 | MA77 Linear Solver | 797 |
| 4.11 | MA86 Linear Solver | 798 |
| 4.12 | MA97 Linear Solver | 798 |
| 4.13 | Mumps Linear Solver | 798 |
| 4.14 | NLP | 798 |
| 4.15 | NLP Scaling | 799 |
| 4.16 | Output | 799 |
| 4.17 | Pardiso Linear Solver | 799 |
| 4.18 | Restoration Phase | 800 |
| 4.19 | Step Calculation | 800 |
| 4.20 | Warm Start | 801 |
| 5 | Detailed Options Description | 801 |
| Bibliography | 829 |
1 The linear solver in IPOPT

The performance and robustness of IPOPT on larger models heavily relies on the used solver for sparse symmetric indefinite linear systems.

GAMS/IPOPT includes the sparse solver MUMPS [1, 2] (currently the default), and MKL PARDISO [7, 8] (only Linux and Windows). In the commercially licensed GAMS/IPOPTH version, also the Harwell Subroutine Library (HSL) solvers MA27, MA57, HSL_MA86, and HSL_MA97 are available and MA27 is used by default.

MUMPS, MA57, HSL_MA86, and HSL_MA97 use METIS for matrix ordering [4], see also the METIS manual. METIS is copyrighted by the regents of the University of Minnesota.

IPOPT and IPOPTH can exploit parallelization of the linear solver or the linear algebra routines (Blas and Lapack). The following table summarizes which options are available on which platform.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Linear Algebra</th>
<th>MUMPS</th>
<th>MKL PARDISO</th>
<th>MA27</th>
<th>MA57</th>
<th>HSL_MA86</th>
<th>HSL_MA97</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>parallel</td>
<td>serial</td>
<td>parallel</td>
<td>serial</td>
<td>serial</td>
<td>parallel</td>
<td>parallel</td>
</tr>
<tr>
<td>MacOS X</td>
<td>parallel</td>
<td>serial</td>
<td>not available</td>
<td>serial</td>
<td>serial</td>
<td>parallel</td>
<td>parallel</td>
</tr>
<tr>
<td>Solaris</td>
<td>serial</td>
<td>serial</td>
<td>not available</td>
<td>serial</td>
<td>serial</td>
<td>parallel</td>
<td>parallel</td>
</tr>
<tr>
<td>Windows</td>
<td>parallel</td>
<td>serial</td>
<td>parallel</td>
<td>serial</td>
<td>serial</td>
<td>parallel</td>
<td>parallel</td>
</tr>
</tbody>
</table>

The linear solver is chosen by the linear_solver option. Benchmarks have shown that MA57 and HSL_MA97 are often able to outperform MA27 on larger instances. Further, PARDISO often allows for performance that is better than MUMPS and similar to the HSL solvers. If IPOPT fails to solve an instance with PARDISO, it’s worth to try changing the options pardiso_order and pardiso_max_iterative_refinement_steps.

2 Usage

The following statement can be used inside your GAMS program to specify using IPOPT

```
Option NLP = IPOPT; { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

The above statement should appear before the Solve statement. If IPOPT was specified as the default solver during GAMS installation, the above statement is not necessary.

To use IPOPTH, the statement should be

```
Option NLP = IPOPTH; { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

2.1 Using Harwell Subroutine Library routines with GAMS/IPOPT.

GAMS/IPOPT can use the HSL routines MA27, MA28, MA57, HSL_MA77, HSL_MA86, HSL_MA97, MC19, and HSL_MC68 when provided as shared library. By telling IPOPT to use one of these routines (see options linear_solver, linear_system_scaling, nlp_scaling_method, dependency_detector), GAMS/IPOPT attempts to load the required routines from the library libhsl.so (Unix-Systems), libhsl.dylib (MacOS X), or libhsl.dll (Windows), respectively.

The HSL routines are available at http://www.hsl.rl.ac.uk/ipopt. Note that it is your responsibility to ensure that you are entitled to download and use these routines!

2.2 Specification of Options

IPOPT has many options that can be adjusted for the algorithm (see Section List of IPOPT Options). Options are all identified by a string name, and their values can be of one of three types: Number (real), Integer, or String. Number options are used
for things like tolerances, integer options are used for things like maximum number of iterations, and string options are used for setting algorithm details, like the NLP scaling method. Options can be set by creating a \texttt{ipopt.opt} file in the directory you are executing IPOPT.

The \texttt{ipopt.opt} file is read line by line and each line should contain the option name, followed by whitespace, and then the value. Comments can be included with the \# symbol. For example,

\begin{verbatim}
# This is a comment

# Turn off the NLP scaling
nlp_scaling_method none

# Change the initial barrier parameter
mu_init 1e-2

# Set the max number of iterations
max_iter 500
\end{verbatim}

is a valid \texttt{ipopt.opt} file.

GAMS/IPOPT understand currently the following GAMS parameters: \texttt{reslim} (time limit), \texttt{iterlim} (iteration limit), \texttt{domlim} (domain violation limit). You can set them either on the command line, e.g. \texttt{iterlim=500}, or inside your GAMS program, e.g. \texttt{Option iterlim=500;}. Further the option \texttt{threads} can be used to control the number of threads used in the linear algebra routines and the linear solver, see also Section \textit{The linear solver in Ipopt}.

## 2.3 Warmstarting Ipopt

As an interior point solver, it is difficult to warm start IPOPT. By default, only the level values of the variables are passed as starting point to IPOPT. Setting the IPOPT option \texttt{warm_start_init_point} to \texttt{yes} enables that also dual values for variables and constraints are passed to IPOPT.

However, the expected behavior that IPOPT finishes within one iteration if optimal primal and dual values are passed is not reached this way, yet. This is, because IPOPT by default moves any initial value that is close to a bound into the interior. The amount on how much the initial point is moved can be controlled by various \texttt{bound_push} and \texttt{bound_frac} options. To make IPOPT accept an optimal primal/dual solution within one iteration, it should be sufficient to set the following options:

\begin{verbatim}
warm_start_init_point yes
warm_start_bound_push 1e-9
warm_start_bound_frac 1e-9
warm_start_slack_bound_frac 1e-9
warm_start_slack_bound_push 1e-9
warm_start_mult_bound_push 1e-9
\end{verbatim}

## 3 Output

This section describes the standard IPOPT console output. The output is designed to provide a quick summary of each iteration as IPOPT solves the problem.

Before IPOPT starts to solve the problem, it displays the problem statistics (number of nonzero-elements in the matrices, number of variables, etc.). Note that if you have fixed variables (both upper and lower bounds are equal), IPOPT may remove these variables from the problem internally and not include them in the problem statistics.

Following the problem statistics, IPOPT will begin to solve the problem and you will see output resembling the following:

\begin{verbatim}
iter  objective  inf_pr  inf_du  lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
0  1.6109693e+01  1.12e+01  5.28e-01  0.0  0.00e+00  -  0.00e+00  0.00e+00  0
1  1.8029749e+01  9.90e-01  6.62e+01  0.1  2.14e-01  2.14e-01  1.00e+00f  1
\end{verbatim}
and the columns of output are defined as

- **item**
  - The current iteration count. This includes regular iterations and iterations while in restoration phase. If the algorithm is in the restoration phase, the letter r will be appended to the iteration number.

- **objective**
  - The unscaled objective value at the current point. During the restoration phase, this value remains the unscaled objective value for the original problem.

- **inf_pr**
  - The unscaled constraint violation at the current point. This quantity is the infinity-norm (max) of the (unscaled) constraint violation. During the restoration phase, this value remains the constraint violation of the original problem at the current point. The option inf_pr_output can be used to switch to the printing of a different quantity. During the restoration phase, this value is the primal infeasibility of the original problem at the current point.

- **inf_du**
  - The scaled dual infeasibility at the current point. This quantity measure the infinity-norm (max) of the internal dual infeasibility (Eq. (4a) in [12]), including inequality constraints reformulated using slack variables and problem scaling. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem.

- **lg(mu)**
  - \( \log_{10} \) of the value of the barrier parameter \( \mu \).

- **||d||**
  - The infinity norm (max) of the primal step (for the original variables \( x \) and the internal slack variables \( s \)). During the restoration phase, this value includes the values of additional variables, \( p \) and \( n \) in Eq. (10) of [12].

- **lg(rg)**
  - \( \log_{10} \) of the value of the regularization term for the Hessian of the Lagrangian in the augmented system (\( \delta_w \) in Eq. (26) of [12]). A dash (\( - \)) indicates that no regularization was done.

- **alpha_du**
  - The stepsize for the dual variables (\( \alpha_z^k \) in Eq. (14c) of [12]).

- **alpha_pr**
  - The stepsize for the primal variables (\( \alpha_p \) in Eq. (14a) of [12]). The number is usually followed by a character for additional diagnostic information regarding the step acceptance criterion:
    - f: f-type iteration in the filter method w/o second order correction
    - F: f-type iteration in the filter method w/ second order correction
    - h: h-type iteration in the filter method w/o second order correction
    - H: h-type iteration in the filter method w/ second order correction
    - k: penalty value unchanged in merit function method w/o second order correction
    - K: penalty value unchanged in merit function method w/ second order correction
    - n: penalty value updated in merit function method w/o second order correction
    - N: penalty value updated in merit function method w/ second order correction
    - R: Restoration phase just started
    - w: in watchdog procedure
    - s: step accepted in soft restoration phase
    - t/T: tiny step accepted without line search
- *r*: some previous iterate restored

ls

The number of backtracking line search steps (does not include second-order correction steps).

Note that the step acceptance mechanisms in IPOPT consider the barrier objective function (Eq. (3a) in [12]) which is usually different from the value reported in the objective column. Similarly, for the purposes of the step acceptance, the constraint violation is measured for the internal problem formulation, which includes slack variables for inequality constraints and potentially scaling of the constraint functions. This value, too, is usually different from the value reported in inf_pr. As a consequence, a new iterate might have worse values both for the objective function and the constraint violation as reported in the iteration output, seemingly contradicting globalization procedure.

When the algorithm terminates, IPOPT will output a message to the screen. The following is a list of the possible output messages and a brief description.

Optimal Solution Found.

This message indicates that IPOPT found a (locally) optimal point within the desired tolerances.

Solved To Acceptable Level.

This indicates that the algorithm did not converge to the "desired" tolerances, but that it was able to obtain a point satisfying the "acceptable" tolerance level as specified by acceptable-* options. This may happen if the desired tolerances are too small for the current problem.

Feasible point for square problem found.

This message is printed if the problem is "square" (i.e., it has as many equality constraints as free variables) and IPOPT found a feasible point.

Converged to a point of local infeasibility. Problem may be infeasible.

The restoration phase converged to a point that is a minimizer for the constraint violation (in the $\ell_1$-norm), but is not feasible for the original problem. This indicates that the problem may be infeasible (or at least that the algorithm is stuck at a locally infeasible point). The returned point (the minimizer of the constraint violation) might help you to find which constraint is causing the problem. If you believe that the NLP is feasible, it might help to start the optimization from a different point.

Search Direction is becoming Too Small.

This indicates that IPOPT is calculating very small step sizes and making very little progress. This could happen if the problem has been solved to the best numerical accuracy possible given the current scaling.

Iterates diverging; problem might be unbounded.

This message is printed if the max-norm of the iterates becomes larger than the value of the option diverging_iterates_tol. This can happen if the problem is unbounded below and the iterates are diverging.

Stopping optimization at current point as requested by user.

This message is printed if either the Ctrl+C was pressed or the domain violation limit is reached.

Maximum Number of Iterations Exceeded.

This indicates that IPOPT has exceeded the maximum number of iterations as specified by the IPOPT option max_iter or the GAMS option iterlim.

Maximum CPU time exceeded.

This indicates that IPOPT has exceeded the maximum number of seconds as specified by the IPOPT option max_cpu_time or the GAMS option reslim.

Restoration Failed!

This indicates that the restoration phase failed to find a feasible point that was acceptable to the filter line search for the original problem. This could happen if the problem is highly degenerate or does not satisfy the constraint qualification, or if an external function in GAMS provides incorrect derivative information.

Error in step computation (regularization becomes too large?)!
This message is printed if IPOPT is unable to compute a search direction, despite several attempts to modify the iteration matrix. Usually, the value of the regularization parameter then becomes too large.

Problem has too few degrees of freedom.

This indicates that your problem, as specified, has too few degrees of freedom. This can happen if you have too many equality constraints, or if you fix too many variables (IPOPT removes fixed variables).

Not enough memory.

An error occurred while trying to allocate memory. The problem may be too large for your current memory and swap configuration.

INTERNAL ERROR: Unknown SolverReturn value - Notify IPOPT Authors.

An unknown internal error has occurred. Please notify the authors of the GAMS/IPOPT link or IPOPT (refer to support@gams.com).

3.1 Diagnostic Tags for IPOPT

To print additional diagnostic tags for each iteration of IPOPT, set the options print_info_string to yes. With this, a tag will appear at the end of an iteration line with the following diagnostic meaning that are useful to flag difficulties for a particular IPOPT run. The following is a list of possible strings:

- !: Tighten resto tolerance if only slightly infeasible, see Sec. 3.3 in [12]
- A: Current iteration is acceptable (alternate termination)
- a: Perturbation for PD Singularity can’t be done, assume singular, see Sec. 3.1 in [12]
- C: Second Order Correction taken, see Sec. 2.4 in [12]
- Dh: Hessian degenerate based on multiple iterations, see Sec. 3.1 in [12]
- Dhj: Hessian/Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [12]
- Dj: Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [12]
- dx: δx perturbation too large, see Sec. 3.1 in [12]
- e: Cutting back α due to evaluation error (in backtracking line search)
- F-: Filter should be reset, but maximal resets exceeded, see Sec. 2.3 in [12]
- F+: Resetting filter due to last few rejections of filter, see Sec. 2.3 in [12]
- L: Degenerate Jacobian, δc already perturbed, see Sec. 3.1 in [12]
- l: Degenerate Jacobian, δc perturbed, see Sec. 3.1 in [12]
- M: Magic step taken for slack variables (in backtracking line search)
- Nh: Hessian not yet degenerate, see Sec. 3.1 in [12]
- Nhj: Hessian/Jacobian not yet degenerate, see Sec. 3.1 in [12]
- Nj: Jacobian not yet degenerate, see Sec. 3.1 in [12]
- Nw: Warm start initialization failed (in Warm Start Initialization)
- q: PD system possibly singular, attempt to improve solution quality, see Sec. 3.1 in [12]
- R: Solution of restoration phase, see Sec. 3.3 in [12]
- S: PD system possibly singular, accept current solution, see Sec. 3.1 in [12]
• s: PD system singular, see Sec. 3.1 in [12]

• s: Square Problem. Set multipliers to zero (default initialization routine)

• Tmax: Trial $\theta$ is larger than $\theta_{\text{max}}$ (filter parameter, Eq. (21) in [12])

• W: Watchdog line search procedure successful, see Sec. 3.2 in [12]

• w: Watchdog line search procedure unsuccessful, stopped, see Sec. 3.2 in [12]

• Wb: Undoing most recent SR1 update, see Sec. 5.4.1 in [3]

• We: Skip Limited-Memory Update in restoration phase, see Sec. 5.4.1 in [3]

• Wp: Safeguard $B^0 = \sigma I$ for Limited-Memory Update, see Sec. 5.4.1 in [3]

• Wr: Resetting Limited-Memory Update, see Sec. 5.4.1 in [3]

•Ws: Skip Limited-Memory Update since $s^Ty$ is not positive, see Sec. 5.4.1 in [3]

• WS: Skip Limited-Memory Update since $\Delta x$ is too small, see Sec. 5.4.1 in [3]

• y: Dual infeasibility, use least square multiplier update (during IPOPT algorithm)

• z: Apply correction to bound multiplier if too large (during IPOPT algorithm)

4 List of IPOPT Options

4.1 Barrier Parameter Update

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptive_mu_globalization</td>
<td>Globalization strategy for the adaptive mu selection mode.</td>
<td>obj-constr-filter</td>
</tr>
<tr>
<td>adaptive_mu_kkterror_red_fact</td>
<td>Sufficient decrease factor for 'kkt-error' globalization strategy.</td>
<td>0.9999</td>
</tr>
<tr>
<td>adaptive_mu_kkterror_red_iters</td>
<td>Maximum number of iterations requiring sufficient progress.</td>
<td>4</td>
</tr>
<tr>
<td>adaptive_mu_kkt_norm_type</td>
<td>Norm used for the KKT error in the adaptive mu globalization strategies.</td>
<td>2-norm-squared</td>
</tr>
<tr>
<td>adaptive_mu_monotone_init_factor</td>
<td>Determines the initial value of the barrier parameter when switching to the monotone mode.</td>
<td>0.8</td>
</tr>
<tr>
<td>adaptive_mu_restore_previous_iterate</td>
<td>Indicates if the previous iterate should be restored if the monotone mode is entered.</td>
<td>no</td>
</tr>
<tr>
<td>barrier_tol_factor</td>
<td>Factor for mu in barrier stop test.</td>
<td>10</td>
</tr>
<tr>
<td>filter_margin_fact</td>
<td>Factor determining width of margin for obj-constr-filter adaptive globalization strategy.</td>
<td>1e-05</td>
</tr>
<tr>
<td>filter_max_margin</td>
<td>Maximum width of margin in obj-constr-filter adaptive globalization strategy.</td>
<td>1</td>
</tr>
<tr>
<td>fixed_mu_oracle</td>
<td>Oracle for the barrier parameter when switching to fixed mode.</td>
<td>average_compl</td>
</tr>
<tr>
<td>mu_allow_fast_monotone_decrease</td>
<td>Allow skipping of barrier problem if barrier test is already met.</td>
<td>yes</td>
</tr>
<tr>
<td>mu_init</td>
<td>Initial value for the barrier parameter.</td>
<td>0.1</td>
</tr>
<tr>
<td>mu_linear_decrease_factor</td>
<td>Determines linear decrease rate of barrier parameter.</td>
<td>0.2</td>
</tr>
<tr>
<td>mu_max</td>
<td>Maximum value for barrier parameter.</td>
<td>100000</td>
</tr>
</tbody>
</table>
### 4.2 Convergence

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptable_compl_inf_tol</td>
<td>'Acceptance' threshold for the complementarity conditions.</td>
<td>0.01</td>
</tr>
<tr>
<td>acceptable_constr_viol_tol</td>
<td>'Acceptance' threshold for the constraint violation.</td>
<td>0.01</td>
</tr>
<tr>
<td>acceptable_dual_inf_tol</td>
<td>'Acceptance' threshold for the dual infeasibility.</td>
<td>1e+10</td>
</tr>
<tr>
<td>acceptable_obj_change_tol</td>
<td>'Acceptance' stopping criterion based on objective function change.</td>
<td>1e+20</td>
</tr>
<tr>
<td>acceptable_tol</td>
<td>'Acceptable' convergence tolerance (relative).</td>
<td>1e-06</td>
</tr>
<tr>
<td>compl_inf_tol</td>
<td>Desired threshold for the complementarity conditions.</td>
<td>0.0001</td>
</tr>
<tr>
<td>constr_viol_tol</td>
<td>Desired threshold for the constraint violation.</td>
<td>0.0001</td>
</tr>
<tr>
<td>diverging_iterates_tol</td>
<td>Threshold for maximal value of primal iterates.</td>
<td>1e+20</td>
</tr>
<tr>
<td>dual_inf_tol</td>
<td>Desired threshold for the dual infeasibility.</td>
<td>1</td>
</tr>
<tr>
<td>max_cpu_time</td>
<td>Maximum number of CPU seconds.</td>
<td>1000</td>
</tr>
<tr>
<td>max_iter</td>
<td>Maximum number of iterations.</td>
<td>maxint</td>
</tr>
<tr>
<td>mu_target</td>
<td>Desired value of complementarity.</td>
<td>0</td>
</tr>
<tr>
<td>s_max</td>
<td>Scaling threshold for the NLP error.</td>
<td>100</td>
</tr>
<tr>
<td>tol</td>
<td>Desired convergence tolerance (relative).</td>
<td>1e-08</td>
</tr>
</tbody>
</table>

### 4.3 Hessian Approximation
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hessian_approximation</td>
<td>Indicates what Hessian information is to be used.</td>
<td>exact</td>
</tr>
<tr>
<td>hessian_approximation_space</td>
<td>Indicates in which subspace the Hessian information is to be approximated.</td>
<td>nonlinear-variables</td>
</tr>
<tr>
<td>limited_memory_aug_solver</td>
<td>Strategy for solving the augmented system for low-rank Hessian.</td>
<td>sherman-morrison</td>
</tr>
<tr>
<td>limited_memory_initialization</td>
<td>Initialization strategy for the limited memory quasi-Newton approximation.</td>
<td>scalar1</td>
</tr>
<tr>
<td>limited_memory_init_val</td>
<td>Value for B0 in low-rank update.</td>
<td>1</td>
</tr>
<tr>
<td>limited_memory_init_val_max</td>
<td>Upper bound on value for B0 in low-rank update.</td>
<td>1e+08</td>
</tr>
<tr>
<td>limited_memory_init_val_min</td>
<td>Lower bound on value for B0 in low-rank update.</td>
<td>1e-08</td>
</tr>
<tr>
<td>limited_memory_max_history</td>
<td>Maximum size of the history for the limited quasi-Newton Hessian approximation.</td>
<td>6</td>
</tr>
<tr>
<td>limited_memory_max_skipping</td>
<td>Threshold for successive iterations where update is skipped.</td>
<td>2</td>
</tr>
<tr>
<td>limited_memory_special_for_resto</td>
<td>Determines if the quasi-Newton updates should be special during the restoration phase.</td>
<td>no</td>
</tr>
<tr>
<td>limited_memory_update_type</td>
<td>Quasi-Newton update formula for the limited memory approximation.</td>
<td>bfgs</td>
</tr>
</tbody>
</table>

### 4.4 Initialization

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound_frac</td>
<td>Desired minimum relative distance from the initial point to bound.</td>
<td>0.01</td>
</tr>
<tr>
<td>bound_mult_init_method</td>
<td>Initialization method for bound multipliers</td>
<td>constant</td>
</tr>
<tr>
<td>bound_mult_init_val</td>
<td>Initial value for the bound multipliers.</td>
<td>1</td>
</tr>
<tr>
<td>bound_push</td>
<td>Desired minimum absolute distance from the initial point to bound.</td>
<td>0.01</td>
</tr>
<tr>
<td>constr_mult_init_max</td>
<td>Maximum allowed least-square guess of constraint multipliers.</td>
<td>1000</td>
</tr>
<tr>
<td>least_square_init_duals</td>
<td>Least square initialization of all dual variables</td>
<td>no</td>
</tr>
<tr>
<td>least_square_init_primal</td>
<td>Least square initialization of the primal variables</td>
<td>no</td>
</tr>
<tr>
<td>slack_bound_frac</td>
<td>Desired minimum relative distance from the initial slack to bound.</td>
<td>0.01</td>
</tr>
<tr>
<td>slack_bound_push</td>
<td>Desired minimum absolute distance from the initial slack to bound.</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### 4.5 Line Search

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept_after_max_steps</td>
<td>Accept a trial point after maximal this number of steps.</td>
<td>-1</td>
</tr>
<tr>
<td>accept_every_trial_step</td>
<td>Always accept the first trial step.</td>
<td>no</td>
</tr>
<tr>
<td>alpha_for_y</td>
<td>Method to determine the step size for constraint multipliers.</td>
<td>primal</td>
</tr>
<tr>
<td>alpha_for_y_tol</td>
<td>Tolerance for switching to full equality multiplier steps.</td>
<td>10</td>
</tr>
<tr>
<td>alpha_min_frac</td>
<td>Safety factor for the minimal step size (before switching to restoration phase).</td>
<td>0.05</td>
</tr>
<tr>
<td>alpha_red_factor</td>
<td>Fractional reduction of the trial step size in the backtracking line search.</td>
<td>0.5</td>
</tr>
<tr>
<td>constraintViolation_norm_type</td>
<td>Norm to be used for the constraint violation in the line search.</td>
<td>1-norm</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>corrector_compl_avrg_red_fact</code></td>
<td>Complementarity tolerance factor for accepting corrector step (unsupported!).</td>
<td>1</td>
</tr>
<tr>
<td><code>corrector_type</code></td>
<td>The type of corrector steps that should be taken (unsupported!).</td>
<td><code>none</code></td>
</tr>
<tr>
<td><code>delta</code></td>
<td>Multiplier for constraint violation in the switching rule.</td>
<td>1</td>
</tr>
<tr>
<td><code>eta_phi</code></td>
<td>Relaxation factor in the Armijo condition.</td>
<td><code>1e-08</code></td>
</tr>
<tr>
<td><code>filter_reset_trigger</code></td>
<td>Number of iterations that trigger the filter reset.</td>
<td>5</td>
</tr>
<tr>
<td><code>gamma_phi</code></td>
<td>Relaxation factor in the filter margin for the filter function.</td>
<td><code>1e-08</code></td>
</tr>
<tr>
<td><code>gamma_theta</code></td>
<td>Relaxation factor in the filter margin for the constraint violation.</td>
<td><code>1e-05</code></td>
</tr>
<tr>
<td><code>kappa_soc</code></td>
<td>Factor limiting the deviation of dual variables from primal estimates.</td>
<td><code>1e+10</code></td>
</tr>
<tr>
<td><code>line_search_method</code></td>
<td>Globalization method used in backtracking line search</td>
<td><code>filter</code></td>
</tr>
<tr>
<td><code>max_filter_resets</code></td>
<td>Maximal allowed number of filter resets</td>
<td>5</td>
</tr>
<tr>
<td><code>max_soc</code></td>
<td>Maximum number of second order correction trial steps at each iteration.</td>
<td>4</td>
</tr>
<tr>
<td><code>nu_inc</code></td>
<td>Increment of the penalty parameter.</td>
<td><code>0.0001</code></td>
</tr>
<tr>
<td><code>nu_init</code></td>
<td>Initial value of the penalty parameter.</td>
<td><code>1e-06</code></td>
</tr>
<tr>
<td><code>obj_max_inc</code></td>
<td>Determines the upper bound on the acceptable increase of barrier objective function.</td>
<td>5</td>
</tr>
<tr>
<td><code>recalc_y</code></td>
<td>Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.</td>
<td><code>no</code></td>
</tr>
<tr>
<td><code>recalc_y_feas_tol</code></td>
<td>Feasibility threshold for recomputation of multipliers.</td>
<td><code>1e-06</code></td>
</tr>
<tr>
<td><code>rho</code></td>
<td>Value in penalty parameter update formula.</td>
<td>0.1</td>
</tr>
<tr>
<td><code>skip_corr_if_neg_curv</code></td>
<td>Skip the corrector step in negative curvature iteration (unsupported!).</td>
<td><code>yes</code></td>
</tr>
<tr>
<td><code>skip_corr_in_monotone_mode</code></td>
<td>Skip the corrector step during monotone barrier parameter mode (unsupported!).</td>
<td><code>yes</code></td>
</tr>
<tr>
<td><code>slack_move</code></td>
<td>Correction size for very small slacks.</td>
<td><code>1.81899e-12</code></td>
</tr>
<tr>
<td><code>s_phi</code></td>
<td>Exponent for linear barrier function model in the switching rule.</td>
<td>2.3</td>
</tr>
<tr>
<td><code>s_theta</code></td>
<td>Exponent for current constraint violation in the switching rule.</td>
<td>1.1</td>
</tr>
<tr>
<td><code>theta_max_fact</code></td>
<td>Determines upper bound for constraint violation in the filter.</td>
<td>10000</td>
</tr>
<tr>
<td><code>theta_min_fact</code></td>
<td>Determines constraint violation threshold in the switching rule.</td>
<td>0.0001</td>
</tr>
<tr>
<td><code>tiny_step_tol</code></td>
<td>Tolerance for detecting numerically insignificant steps.</td>
<td><code>2.22045e-15</code></td>
</tr>
<tr>
<td><code>tiny_step_y_tol</code></td>
<td>Tolerance for quitting because of numerically insignificant steps.</td>
<td>0.01</td>
</tr>
<tr>
<td><code>watchdog_shortened_iter_trigger</code></td>
<td>Number of shortened iterations that trigger the watchdog.</td>
<td>10</td>
</tr>
<tr>
<td><code>watchdog_trial_iter_max</code></td>
<td>Maximum number of watchdog iterations.</td>
<td>3</td>
</tr>
</tbody>
</table>

### 4.6 Linear Solver

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>linear_scaling_on_demand</code></td>
<td>Flag indicating that linear scaling is only done if it seems required.</td>
<td><code>yes</code></td>
</tr>
<tr>
<td><code>linear_solver</code></td>
<td>Linear solver used for step computations.</td>
<td><code>ma27</code></td>
</tr>
<tr>
<td><code>linear_system_scaling</code></td>
<td>Method for scaling the linear system.</td>
<td><code>mc19</code></td>
</tr>
</tbody>
</table>

### 4.7 MA27 Linear Solver
### 4.8 MA28 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma28_pivtol</td>
<td>Pivot tolerance for linear solver MA28.</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### 4.9 MA57 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma57_automatic_scaling</td>
<td>Controls MA57 automatic scaling</td>
<td>no</td>
</tr>
<tr>
<td>ma57_block_size</td>
<td>Controls block size used by Level 3 BLAS in MA57BD</td>
<td>16</td>
</tr>
<tr>
<td>ma57_node_amalgamation</td>
<td>Node amalgamation parameter</td>
<td>16</td>
</tr>
<tr>
<td>ma57_pivot_order</td>
<td>Controls pivot order in MA57</td>
<td>5</td>
</tr>
<tr>
<td>ma57_pivtol</td>
<td>Pivot tolerance for the linear solver MA57</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma57_pivtolmax</td>
<td>Maximum pivot tolerance for the linear solver MA57</td>
<td>0.0001</td>
</tr>
<tr>
<td>ma57_pre_alloc</td>
<td>Safety factor for work space memory allocation for the linear solver MA57.</td>
<td>1.05</td>
</tr>
<tr>
<td>ma57_small_pivot_flag</td>
<td>If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.</td>
<td>0</td>
</tr>
</tbody>
</table>

### 4.10 MA77 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma77_buffer_lpage</td>
<td>Number of scalars per MA77 buffer page</td>
<td>4096</td>
</tr>
<tr>
<td>ma77_buffer_npage</td>
<td>Number of pages that make up MA77 buffer</td>
<td>1600</td>
</tr>
<tr>
<td>ma77_file_size</td>
<td>Target size of each temporary file for MA77, scalars per type</td>
<td>2097152</td>
</tr>
<tr>
<td>ma77_maxstore</td>
<td>Maximum storage size for MA77 in-core mode</td>
<td>0</td>
</tr>
<tr>
<td>ma77_nemin</td>
<td>Node Amalgamation parameter</td>
<td>8</td>
</tr>
<tr>
<td>ma77_order</td>
<td>Controls type of ordering used by HSL_MAM77</td>
<td>metis</td>
</tr>
<tr>
<td>ma77_print_level</td>
<td>Debug printing level for the linear solver MA77</td>
<td>-1</td>
</tr>
<tr>
<td>ma77_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma77_static</td>
<td>Static Pivoting Threshold</td>
<td>0</td>
</tr>
<tr>
<td>ma77_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma77_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>
## 4.11 MA86 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma86_nemin</td>
<td>Node Amalgamation parameter</td>
<td>32</td>
</tr>
<tr>
<td>ma86_order</td>
<td>Controls type of ordering used by HSL_MA86</td>
<td>auto</td>
</tr>
<tr>
<td>ma86_print_level</td>
<td>Debug printing level for the linear solver MA86</td>
<td>-1</td>
</tr>
<tr>
<td>ma86_scaling</td>
<td>Controls scaling of matrix</td>
<td>mc64</td>
</tr>
<tr>
<td>ma86_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma86_static</td>
<td>Static Pivoting Threshold</td>
<td>0</td>
</tr>
<tr>
<td>ma86_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma86_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

## 4.12 MA97 Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ma97_nemin</td>
<td>Node Amalgamation parameter</td>
<td>8</td>
</tr>
<tr>
<td>ma97_order</td>
<td>Controls type of ordering used by HSL_MA97</td>
<td>auto</td>
</tr>
<tr>
<td>ma97_print_level</td>
<td>Debug printing level for the linear solver MA97</td>
<td>0</td>
</tr>
<tr>
<td>ma97_scaling</td>
<td>Specifies strategy for scaling in HSL_MA97 linear solver</td>
<td>dynamic</td>
</tr>
<tr>
<td>ma97_scaling1</td>
<td>First scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_scaling2</td>
<td>Second scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_scaling3</td>
<td>Third scaling.</td>
<td>mc64</td>
</tr>
<tr>
<td>ma97_small</td>
<td>Zero Pivot Threshold</td>
<td>1e-20</td>
</tr>
<tr>
<td>ma97_solve_blas3</td>
<td>Controls if blas2 or blas3 routines are used for solve</td>
<td>no</td>
</tr>
<tr>
<td>ma97_switch1</td>
<td>First switch, determine when ma97_scaling1 is enabled.</td>
<td>od_hd_reuse</td>
</tr>
<tr>
<td>ma97_switch2</td>
<td>Second switch, determine when ma97_scaling2 is enabled.</td>
<td>never</td>
</tr>
<tr>
<td>ma97_switch3</td>
<td>Third switch, determine when ma97_scaling3 is enabled.</td>
<td>never</td>
</tr>
<tr>
<td>ma97_u</td>
<td>Pivoting Threshold</td>
<td>1e-08</td>
</tr>
<tr>
<td>ma97_umax</td>
<td>Maximum Pivoting Threshold</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

## 4.13 Mumps Linear Solver

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mumps_dep_tol</td>
<td>Pivot threshold for detection of linearly dependent constraints in MUMPS.</td>
<td>0</td>
</tr>
<tr>
<td>mumps_mem_percent</td>
<td>Percentage increase in the estimated working space for MUMPS.</td>
<td>1000</td>
</tr>
<tr>
<td>mumps_permuting_scaling</td>
<td>Controls permuting and scaling in MUMPS</td>
<td>7</td>
</tr>
<tr>
<td>mumps_pivot_order</td>
<td>Controls pivot order in MUMPS</td>
<td>7</td>
</tr>
<tr>
<td>mumps_pivotolmax</td>
<td>Maximum pivot tolerance for the linear solver MUMPS.</td>
<td>1e-06</td>
</tr>
<tr>
<td>mumps_scaling</td>
<td>Controls scaling in MUMPS</td>
<td>77</td>
</tr>
</tbody>
</table>

## 4.14 NLP

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
**4.15 NLP Scaling**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlp_scaling_constr_target_gradient</td>
<td>Target value for constraint function gradient size.</td>
<td>0</td>
</tr>
<tr>
<td>nlp_scaling_max_gradient</td>
<td>Maximum gradient after NLP scaling.</td>
<td>100</td>
</tr>
<tr>
<td>nlp_scaling_method</td>
<td>Select the technique used for scaling the NLP.</td>
<td>gradient-based</td>
</tr>
<tr>
<td>nlp_scaling_min_value</td>
<td>Minimum value of gradient-based scaling values.</td>
<td>1e-08</td>
</tr>
<tr>
<td>nlp_scaling_obj_target_gradient</td>
<td>Target value for objective function gradient size.</td>
<td>0</td>
</tr>
</tbody>
</table>

**4.16 Output**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf_pr_output</td>
<td>Determines what value is printed in the 'inf_pr' output column.</td>
<td>original</td>
</tr>
<tr>
<td>print_eval_error</td>
<td>Switch to enable printing information about function evaluation errors into the GAMS listing file.</td>
<td>yes</td>
</tr>
<tr>
<td>print_frequency_iter</td>
<td>Determines at which iteration frequency the summarizing iteration output line should be printed.</td>
<td>1</td>
</tr>
<tr>
<td>print_frequency_time</td>
<td>Determines at which time frequency the summarizing iteration output line should be printed.</td>
<td>0</td>
</tr>
<tr>
<td>print_info_string</td>
<td>Enables printing of additional info string at end of iteration output.</td>
<td>no</td>
</tr>
<tr>
<td>print_level</td>
<td>Output verbosity level.</td>
<td>5</td>
</tr>
<tr>
<td>print_timing_statistics</td>
<td>Switch to print timing statistics.</td>
<td>no</td>
</tr>
<tr>
<td>replace_bounds</td>
<td>Indicates if all variable bounds should be replaced by inequality constraints</td>
<td>no</td>
</tr>
<tr>
<td>report_mininfeas_solution</td>
<td>Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible.</td>
<td>no</td>
</tr>
</tbody>
</table>

**4.17 Pardiso Linear Solver**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pardiso_matching_strategy</td>
<td>Matching strategy to be used by Pardiso</td>
<td>complete+2x2</td>
</tr>
</tbody>
</table>
### 4.18 Restoration Phase

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound_mult_reset_threshold</td>
<td>Threshold for resetting bound multipliers after the restoration phase.</td>
<td>1000</td>
</tr>
<tr>
<td>constr_mult_reset_threshold</td>
<td>Threshold for resetting equality and inequality multipliers after restoration phase.</td>
<td>0</td>
</tr>
<tr>
<td>evaluate_orig_obj_at_resto_trial</td>
<td>Determines if the original objective function should be evaluated at restoration phase trial points.</td>
<td>yes</td>
</tr>
<tr>
<td>expect_infeasible_problem</td>
<td>Enable heuristics to quickly detect an infeasible problem.</td>
<td>no</td>
</tr>
<tr>
<td>expect_infeasible_problem_ctol</td>
<td>Threshold for disabling 'expect_infeasible_problem' option.</td>
<td>0.001</td>
</tr>
<tr>
<td>expect_infeasible_problem_ytol</td>
<td>Multiplier threshold for activating 'expect_infeasible_problem' option.</td>
<td>1e+08</td>
</tr>
<tr>
<td>max_resto_iter</td>
<td>Maximum number of successive iterations in restoration phase.</td>
<td>3000000</td>
</tr>
<tr>
<td>max_soft_resto_iters</td>
<td>Maximum number of iterations performed successively in soft restoration phase.</td>
<td>10</td>
</tr>
<tr>
<td>required_infeasibility_reduction</td>
<td>Required reduction of infeasibility before leaving restoration phase.</td>
<td>0.9</td>
</tr>
<tr>
<td>resto_failure_feasibility_threshold</td>
<td>Threshold for primal infeasibility to declare failure of restoration phase.</td>
<td>0</td>
</tr>
<tr>
<td>resto_penalty_parameter</td>
<td>Penalty parameter in the restoration phase objective function.</td>
<td>1000</td>
</tr>
<tr>
<td>resto_proximity_weight</td>
<td>Weighting factor for the proximity term in restoration phase objective.</td>
<td>1</td>
</tr>
<tr>
<td>soft_resto_pderror_reduction_factor</td>
<td>Required reduction in primal-dual error in the soft restoration phase.</td>
<td>0.9999</td>
</tr>
<tr>
<td>start_with_resto</td>
<td>Tells algorithm to switch to restoration phase in first iteration.</td>
<td>no</td>
</tr>
</tbody>
</table>

### 4.19 Step Calculation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast_step_computation</td>
<td>Indicates if the linear system should be solved quickly.</td>
<td>no</td>
</tr>
<tr>
<td>first_hessian_perturbation</td>
<td>Size of first x-s perturbation tried.</td>
<td>0.0001</td>
</tr>
<tr>
<td>jacobian_regularization_exponent</td>
<td>Exponent for mu in the regularization for rank-deficient constraint Jacobians.</td>
<td>0.25</td>
</tr>
<tr>
<td>jacobian_regularization_value</td>
<td>Size of the regularization for rank-deficient constraint Jacobians.</td>
<td>1e-08</td>
</tr>
<tr>
<td>max_hessian_perturbation</td>
<td>Maximum value of regularization parameter for handling negative curvature.</td>
<td>1e+20</td>
</tr>
<tr>
<td>max_refinement_steps</td>
<td>Maximum number of iterative refinement steps per linear system solve.</td>
<td>10</td>
</tr>
<tr>
<td>mehrotra_algorithm</td>
<td>Indicates if we want to do Mehrotra’s algorithm.</td>
<td>no</td>
</tr>
<tr>
<td>min_hessian_perturbation</td>
<td>Smallest perturbation of the Hessian block.</td>
<td>1e-20</td>
</tr>
<tr>
<td>min_refinement_steps</td>
<td>Minimum number of iterative refinement steps per linear system solve.</td>
<td>1</td>
</tr>
<tr>
<td>neg_curv_test_tol</td>
<td>Tolerance for heuristic to ignore wrong inertia.</td>
<td>0</td>
</tr>
</tbody>
</table>
### IPOPT and IPOPTH 801

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>perturb_always_cd</td>
<td>Active permanent perturbation of constraint linearization.</td>
<td>no</td>
</tr>
<tr>
<td>perturb_dec_fact</td>
<td>Decrease factor for x-s perturbation.</td>
<td>0.333333</td>
</tr>
<tr>
<td>perturb_inc_fact</td>
<td>Increase factor for x-s perturbation.</td>
<td>8</td>
</tr>
<tr>
<td>perturb_inc_fact_first</td>
<td>Increase factor for x-s perturbation for very first perturbation.</td>
<td>100</td>
</tr>
<tr>
<td>residual_improvement_factor</td>
<td>Minimal required reduction of residual test ratio in iterative refinement.</td>
<td>1</td>
</tr>
<tr>
<td>residual_ratio_max</td>
<td>Iterative refinement tolerance</td>
<td>1e-10</td>
</tr>
<tr>
<td>residual_ratio_singular</td>
<td>Threshold for declaring linear system singular after failed iterative refinement.</td>
<td>1e-05</td>
</tr>
</tbody>
</table>

#### 4.20 Warm Start

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>warm_start_bound_frac</td>
<td>same as bound_frac for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_bound_push</td>
<td>same as bound_push for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_init_point</td>
<td>Warm-start for initial point</td>
<td>no</td>
</tr>
<tr>
<td>warm_start_mult_bound_push</td>
<td>same as mult_bound_push for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_mult_init_max</td>
<td>Maximum initial value for the equality multipliers.</td>
<td>1e+06</td>
</tr>
<tr>
<td>warm_start_slack_bound_frac</td>
<td>same as slack_bound_frac for the regular initializer.</td>
<td>0.001</td>
</tr>
<tr>
<td>warm_start_slack_bound_push</td>
<td>same as slack_bound_push for the regular initializer.</td>
<td>0.001</td>
</tr>
</tbody>
</table>

#### 5 Detailed Options Description

- **acceptable_compl_inf_tol** *(real)*: 'Acceptance' threshold for the complementarity conditions.
  
  Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also acceptable_tol.
  
  (default = 0.01)

- **acceptable_constr_viol_tol** *(real)*: 'Acceptance' threshold for the constraint violation.
  
  Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also acceptable_tol.
  
  (default = 0.01)

- **acceptable_dual_inf_tol** *(real)*: 'Acceptance' threshold for the dual infeasibility.
  
  Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also acceptable_tol.
  
  (default = 1e+10)

- **acceptable_iter** *(integer)*: Number of 'acceptable' iterates before triggering termination.
  
  If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled.
  
  (default = 15)

- **acceptable_obj_change_tol** *(real)*: 'Acceptance' stopping criterion based on objective function change.
  
  If the relative change of the objective function (scaled by Max(1,|f(x)|)) is less than this value, this part of the acceptable tolerance termination is satisfied; see also acceptable_tol. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility.
  
  (default = 1e+20)
acceptable_tol (real): 'Acceptable' convergence tolerance (relative).

Determine which (scaled) overall optimality error is considered to be "acceptable." There are two levels of termination criteria. If the usual "desired" tolerances (see tol, dual_inf_tol etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the "desired" level of accuracy.

(default = 1e-06)

accept_after_max_steps (integer): Accept a trial point after maximal this number of steps.

Even if it does not satisfy line search conditions.

(default = -1)

accept_every_trial_step (string): Always accept the first trial step.

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees.

(default = no)

no don’t arbitrarily accept the full step

yes always accept the full step

adaptive_mu_globalization (string): Globalization strategy for the adaptive mu selection mode.

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive".)

(default = obj-constr-filter)

kkt-error nonmonotone decrease of kkt-error
never-monotone-mode disables globalization
obj-constr-filter 2-dim filter for objective and constraint violation

adaptive_mu_kkterror_red_fact (real): Sufficient decrease factor for 'kkt-error' globalization strategy.

For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease.

Range: [0, 1]

(default = 0.9999)

adaptive_mu_kkterror_red_iters (integer): Maximum number of iterations requiring sufficient progress.

For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.

(default = 4)

adaptive_mu_kkt_norm_type (string): Norm used for the KKT error in the adaptive mu globalization strategies.

When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this options is also used in the QualityFunctionMuOracle.

(default = 2-norm-squared)

1-norm use the 1-norm (abs sum)
2-norm use 2-norm
2-norm-squared use the 2-norm squared (sum of squares)
max-norm use the infinity norm (max)
IPOPT and IPOPTH

**adaptive_mu_monotone_init_factor (real):** Determines the initial value of the barrier parameter when switching to the monotone mode.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

(default = 0.8)

**adaptive_mu_restore_previous_iterate (string):** Indicates if the previous iterate should be restored if the monotone mode is entered.

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).

(default = no)

  - no don’t restore accepted iterate
  - yes restore accepted iterate

**alpha_for_y (string):** Method to determine the step size for constraint multipliers.

This option determines how the step size (alpha_y) will be calculated when updating the constraint multipliers.

(default = primal)

  - acceptor Call LSAcceptor to get step size for y
  - bound_mult use step size for the bound multipliers (good for LPs)
  - dual_and_full use the dual step size, and full step if delta_x ≤ alpha_for_y_tol
  - full take a full step of size one
  - max use the max of primal and bound multipliers
  - min use the min of primal and bound multipliers
  - min_dual_infeas choose step size minimizing new dual infeasibility
  - primal use primal step size
  - primal_and_full use the primal step size, and full step if delta_x ≤ alpha_for_y_tol
  - safer_min_dual_infeas like 'min_dual_infeas', but safeguarded by 'min' and 'max'

**alpha_for_y_tol (real):** Tolerance for switching to full equality multiplier steps.

This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

(default = 10)

**alpha_min_frac (real):** Safety factor for the minimal step size (before switching to restoration phase).

(This is gamma_alpha in Eqn. (20) in the implementation paper.)

Range: [0, 1]

(default = 0.05)

**alpha_red_factor (real):** Fractional reduction of the trial step size in the backtracking line search.

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: [0, 1]

(default = 0.5)

**barrier_tol_factor (real):** Factor for mu in barrier step test.

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. (This is kappa_epsilon in implementation paper.)
**bound_frac (real):** Desired minimum relative distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: \([0, 0.5]\)

(default = 0.01)

**bound_mult_init_method (string):** Initialization method for bound multipliers

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound_mult_init_val". If "mu-based" is chosen, each value is initialized to the value of \(\mu_{init}\) divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.

(default = constant)

constant set all bound multipliers to the value of bound_mult_init_val

mu-based initialize to \(\mu_{init}/x_{slack}\)

**bound_mult_init_val (real):** Initial value for the bound multipliers.

All dual variables corresponding to bound constraints are initialized to this value.

(default = 1)

**bound_mult_reset_threshold (real):** Threshold for resetting bound multipliers after the restoration phase.

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

(default = 1000)

**bound_push (real):** Desired minimum absolute distance from the initial point to bound.

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

(default = 0.01)

**bound_relax_factor (real):** Factor for initial relaxation of the bounds.

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then bounds relaxation is disabled. (See Eqn.(35) in implementation paper.)

(default = 1e-10)

**check_derivatives_for_naninf (string):** Indicates whether it is desired to check for Nan/Inf in derivative matrices

Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an invalid number is detected, the matrix is written to output with print_level corresponding to J_MORE_DETAILED; so beware of large output!

(default = no)

no Don’t check (faster).

yes Check Jacobians and Hessian for Nan and Inf.

**compl_inf_tol (real):** Desired threshold for the complementarity conditions.

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.

(default = 0.0001)
**constraint violation norm type** *(string)*: Norm to be used for the constraint violation in the line search.

Determines which norm should be used when the algorithm computes the constraint violation in the line search.

(default = 1-norm)

1-norm use the 1-norm
2-norm use the 2-norm
max-norm use the infinity norm

**constr mult init max** *(real)*: Maximum allowed least-square guess of constraint multipliers.

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This option is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero.

(default = 1000)

**constr mult reset threshold** *(real)*: Threshold for resetting equality and inequality multipliers after restoration phase.

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.

(default = 0)

**constr viol tol** *(real)*: Desired threshold for the constraint violation.

Absolute tolerance on the constraint violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold.

(default = 0.0001)

**corrector compl avrg red fact** *(real)*: Complementarity tolerance factor for accepting corrector step (unsupported!).

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted.

(default = 1)

**corrector type** *(string)*: The type of corrector steps that should be taken (unsupported!).

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried.

(default = none)

affine corrector step towards mu=0
none no corrector
primal-dual corrector step towards current mu

**delta** *(real)*: Multiplier for constraint violation in the switching rule.

(See Eqn. (19) in the implementation paper.)

(default = 1)

**dependency detection with rhs** *(string)*: Indicates if the right hand sides of the constraints should be considered during dependency detection

(default = no)

no only look at gradients
yes also consider right hand side

**dependency detector** *(string)*: Indicates which linear solver should be used to detect linearly dependent equality constraints.

The default and available choices depend on how Ipopt has been compiled. This is experimental and does not work well.

(default = none)
ma28 use MA28
mumps use MUMPS
none don’t check; no extra work at beginning

**diverging_iterates_tol (real)**: Threshold for maximal value of primal iterates.
If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.
(default = \(1e+20\))

**dual_inf_tol (real)**: Desired threshold for the dual infeasibility.
Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.
(default = 1)

**eta_phi (real)**: Relaxation factor in the Armijo condition.
(See Eqn. (20) in the implementation paper)
Range: \([0, 0.5]\)
(default = 1e-08)

**evaluate_orig_obj_at_resto_trial (string)**: Determines if the original objective function should be evaluated at restoration phase trial points.
Setting this option to "yes" makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.
(default = yes)
no skip evaluation
yes evaluate at every trial point

**expect_infeasible_problem (string)**: Enable heuristics to quickly detect an infeasible problem.
This option is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.
(default = no)
no the problem probably be feasible
yes the problem has a good chance to be infeasible

**expect_infeasible_problem_ctol (real)**: Threshold for disabling "expect_infeasible_problem" option.
If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this options is set to 0.
(default = 0.001)

**expect_infeasible_problem_ytol (real)**: Multiplier threshold for activating "expect_infeasible_problem" option.
If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.
(default = 1e+08)

**fast_step_computation (string)**: Indicates if the linear system should be solved quickly.
If set to yes, the algorithm assumes that the linear system that is solved to obtain the search direction, is solved sufficiently well. In that case, no residuals are computed, and the computation of the search direction is a little faster.

\( \text{default = no} \)

- no Verify solution of linear system by computing residuals.
- yes Trust that linear systems are solved well.

**filter_margin_fact** *(real):* Factor determining width of margin for obj-constr-filter adaptive globalization strategy.

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: (new obj) < (filter obj) - filter_margin_fact*(new constr-viol) OR (new constr-viol) < (filter constr-viol) - filter_margin_fact*(new constr-viol). For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: \([0, 1]\)

\( \text{default = 1e-05} \)

**filter_max_margin** *(real):* Maximum width of margin in obj-constr-filter adaptive globalization strategy.

\( \text{default = 1} \)

**filter_reset_trigger** *(integer):* Number of iterations that trigger the filter reset.

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

\( \text{default = 5} \)

**first_hessian_perturbation** *(real):* Size of first x-s perturbation tried.

The first value tried for the x-s perturbation in the inertia correction scheme.(This is delta_0 in the implementation paper.)

\( \text{default = 0.0001} \)

**fixed_mu_oracle** *(string):* Oracle for the barrier parameter when switching to fixed mode.

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy").

\( \text{default = average_compl} \)

- average_compl base on current average complementarity
- loqo LOQO’s centrality rule
- probing Mehrotra’s probing heuristic
- quality-function minimize a quality function

**fixed_variable_treatment** *(string):* Determines how fixed variables should be handled.

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to" bound_relax_factor"). For both "make_constraints" and "relax_bounds", bound multipliers are computed for the fixed variables.

\( \text{default = make_parameter} \)

- make_constraint Add equality constraints fixing variables
- make_parameter Remove fixed variable from optimization variables
- relax_bounds Relax fixing bound constraints

**gamma_phi** *(real):* Relaxation factor in the filter margin for the barrier function.

(See Eqn. (18a) in the implementation paper.)
Range: [0, 1]
(default = 1e-08)

**gamma** theta (real): Relaxation factor in the filter margin for the constraint violation.

(See Eqn. (18b) in the implementation paper.)
Range: [0, 1]
(default = 1e-05)

**hessian approximation** (string): Indicates what Hessian information is to be used.
This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.
(default = exact)
- exact Use second derivatives provided by the NLP.
- limited-memory Perform a limited-memory quasi-Newton approximation

**hessian approximation space** (string): Indicates in which subspace the Hessian information is to be approximated.
(default = nonlinear-variables)
- all-variables in space of all variables (without slacks)
- nonlinear-variables only in space of nonlinear variables.

**honor original bounds** (string): Indicates whether final points should be projected into original bounds.
Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization.
(default = yes)
- no Leave final point unchanged
- yes Project final point back into original bounds

**inf pr output** (string): Determines what value is printed in the ‘inf pr’ output column.
Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice “internal” prints out the constraint violation of this formulation. With “original” the true constraint violation in the original NLP is printed.
(default = original)
- internal max-norm of violation of internal equality constraints
- original maximal constraint violation in original NLP

**jacobian regularization exponent** (real): Exponent for mu in the regularization for rank-deficient constraint Jacobians.
(This is kappa_c in the implementation paper.)
(default = 0.25)

**jacobian regularization value** (real): Size of the regularization for rank-deficient constraint Jacobians.
(This is bar delta_c in the implementation paper.)
(default = 1e-08)

**jac c constant** (string): Indicates whether all equality constraints are linear
Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.
(default = no)
- no Don’t assume that all equality constraints are linear
- yes Assume that equality constraints Jacobian are constant
jac_d_constant \textit{(string)}: Indicates whether all inequality constraints are linear

Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

(default = no)

no Don’t assume that all inequality constraints are linear

yes Assume that equality constraints Jacobian are constant

kappa_d \textit{(real)}: Weight for linear damping term (to handle one-sided bounds).

(see Section 3.7 in implementation paper.)

(default = 1e-05)

kappa_sigma \textit{(real)}: Factor limiting the deviation of dual variables from primal estimates.

If the dual variables deviate from their primal estimates, a correction is performed. (See Eqn. (16) in the implementation paper.) Setting the value to less than 1 disables the correction.

(default = 1e+10)

kappa_soc \textit{(real)}: Factor in the sufficient reduction rule for second order correction.

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. (See Step A-5.9 of Algorithm A in the implementation paper.)

(default = 0.99)

least_square_init_duals \textit{(string)}: Least square initialization of all dual variables

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn’t know anything about the starting point, or for solving an LP or QP. This overwrites option “bound_mult_init_method”.

(default = no)

no use bound_mult_init_val and least-square equality constraint multipliers

yes overwrite user-provided point with least-square estimates

least_square_init_primal \textit{(string)}: Least square initialization of the primal variables

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s), to fit the linearized equality and inequality constraints. This might be useful if the user doesn’t know anything about the starting point, or for solving an LP or QP.

(default = no)

no take user-provided point

yes overwrite user-provided point with least-square estimates

limited_memory_aug_solver \textit{(string)}: Strategy for solving the augmented system for low-rank Hessian.

(default = sherman-morrison)

extended use an extended augmented system

sherman-morrison use Sherman-Morrison formula

limited_memory_initialization \textit{(string)}: Initialization strategy for the limited memory quasi-Newton approximation.

Determines how the diagonal Matrix B_0 as the first term in the limited memory approximation should be computed.

(default = scalar1)

constant sigma = limited_memory_init_val

scalar1 sigma = s^T y / s^T s
scalar2 \sigma = y^\top Ty/s^\top Ty

scalar3 arithmetic average of scalar1 and scalar2

scalar4 geometric average of scalar1 and scalar2

**limited_memory_init_val (real):** Value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1)

**limited_memory_init_val_max (real):** Upper bound on value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1e+08)

**limited_memory_init_val_min (real):** Lower bound on value for B0 in low-rank update.

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

(default = 1e-08)

**limited_memory_max_history (integer):** Maximum size of the history for the limited quasi-Newton Hessian approximation.

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.

(default = 6)

**limited_memory_max_skipping (integer):** Threshold for successive iterations where update is skipped.

If the update is skipped more than this number of successive iterations, we quasi-Newton approximation is reset.

(default = 2)

**limited_memory_special_for_resto (string):** Determines if the quasi-Newton updates should be special during the restoration phase.

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

(default = no)

- no use the same update as in regular iterations
- yes use the a special update during restoration phase

**limited_memory_update_type (string):** Quasi-Newton update formula for the limited memory approximation.

Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

(default = bfgs)

- bfgs BFGS update (with skipping)
- sr1 SR1 (not working well)

**linear_scaling_on_demand (string):** Flag indicating that linear scaling is only done if it seems required.

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.
(default = yes)

no Always scale the linear system.

yes Start using linear system scaling if solutions seem not good.

linear_solver (string): Linear solver used for step computations.

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, that MA27, MA57, MA86, and MA97 are only available with a commercially supported GAMS/IpoptH license, or when the user provides a library with HSL code separately. If no GAMS/IpoptH license is available, the default linear solver is MUMPS. Pardiso is only available on Linux and Windows systems. For using Pardiso on non-Linux/Windows systems or MA77, a Pardiso or HSL library need to be provided.

(default = ma27)

ma27 use the Harwell routine MA27
ma57 use the Harwell routine MA57
ma77 use the Harwell routine HSL_MA77
ma86 use the Harwell routine HSL_MA86
ma97 use the Harwell routine HSL_MA97
mumps use MUMPS package
pardiso use the Pardiso package

linear_system_scaling (string): Method for scaling the linear system.

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on-demand" option). This scaling is independent of the NLP problem scaling. By default, MC19 is only used if MA27 or MA57 are selected as linear solvers. Note, that MC19 is only available with a commercially supported GAMS/IpoptH license, or when the user provides a library with HSL code separately. If no commercial GAMS/IpoptH license is available, the default scaling method is slack-based.

(default = mc19)

mc19 use the Harwell routine MC19
none no scaling will be performed
slack-based use the slack values

line_search_method (string): Globalization method used in backtracking line search

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.

(default = filter)

cg-penalty Chen-Goldfarb penalty function
filter Filter method
penalty Standard penalty function

ma27.ignore singularity (string): Enables MA27’s ability to solve a linear system even if the matrix is singular.

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt’s heuristic of small perturbation of the lower diagonal of the KKT matrix.

(default = no)

no Don’t have MA27 solve singular systems
yes Have MA27 solve singular systems

ma27_la_init_factor (real): Real workspace memory for MA27.
The initial real workspace memory = \( \text{la_init_factor} \times \) memory required by unfactored system. Ipopt will increase the workspace size by \( \text{meminc_factor} \) if required. This option is only available if Ipopt has been compiled with MA27.

(default = 5)

**ma27.liw.init.factor** *(real):* Integer workspace memory for MA27.

The initial integer workspace memory = \( \text{liw_init_factor} \times \) memory required by unfactored system. Ipopt will increase the workspace size by \( \text{meminc_factor} \) if required. This option is only available if Ipopt has been compiled with MA27.

(default = 5)

**ma27.meminc.factor** *(real):* Increment factor for workspace size for MA27.

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor. This option is only available if Ipopt has been compiled with MA27.

(default = 2)

**ma27.pivtol** *(real):* Pivot tolerance for the linear solver MA27.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA27.

Range: \([0, 1]\)

(default = 1e-08)

**ma27.pivtolmax** *(real):* Maximum pivot tolerance for the linear solver MA27.

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA27.

Range: \([0, 1]\)

(default = 0.0001)

**ma27.skip.inertia.check** *(string):* Always pretend inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

(default = no)

- no check inertia
- yes skip inertia check

**ma28.pivtol** *(real):* Pivot tolerance for linear solver MA28.

This is used when MA28 tries to find the dependent constraints.

Range: \([0, 1]\)

(default = 0.01)

**ma57.automatic.scaling** *(string):* Controls MA57 automatic scaling

This option controls the internal scaling option of MA57. For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

(default = no)

- no Do not scale the linear system matrix
- yes Scale the linear system matrix

**ma57.block.size** *(integer):* Controls block size used by Level 3 BLAS in MA57BD

This is ICNTL(11) in MA57.

(default = 16)
**ma57_node_amalgamation (integer):** Node amalgamation parameter

This is ICNTL(12) in MA57.

(default = 16)

**ma57_pivot_order (integer):** Controls pivot order in MA57

This is ICNTL(6) in MA57.

Range: [0, 5]

(default = 5)

**ma57_pivot (real):** Pivot tolerance for the linear solver MA57.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA57.

Range: [0, 1]

(default = 1e-08)

**ma57_pivotmax (real):** Maximum pivot tolerance for the linear solver MA57.

Ipopt may increase pivot as high as ma57_pivotmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.

Range: [0, 1]

(default = 0.0001)

**ma57_pre_alloc (real):** Safety factor for work space memory allocation for the linear solver MA57.

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice. This option is only available if Ipopt has been compiled with MA57.

(default = 1.05)

**ma57_small_pivot_flag (integer):** If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.

This is ICNTL(16) in MA57.

Range: [0, 1]

(default = 0)

**ma77_buffer_lpage (integer):** Number of scalars per MA77 buffer page

Number of scalars per an in-core buffer in the out-of-core solver MA77. Must be at most ma77_file_size.

(default = 4096)

**ma77_buffer_npage (integer):** Number of pages that make up MA77 buffer

Number of pages of size buffer_lpage that exist in-core for the out-of-core solver MA77.

(default = 1600)

**ma77_file_size (integer):** Target size of each temporary file for MA77, scalars per type

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.

(default = 2097152)

**ma77_maxstore (integer):** Maximum storage size for MA77 in-core mode

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.

(default = 0)

**ma77_nemin (integer):** Node Amalgamation parameter
Two nodes in elimination tree are merged if result has fewer than ma77_nemin variables.
(default = 8)

**ma77_order** *(string)*: Controls type of ordering used by HSL_MA77

This option controls ordering for the solver HSL_MA77.
(default = metis)

  - amd Use the HSL_MC68 approximate minimum degree algorithm
  - metis Use the MeTiS nested dissection algorithm (if available)

**ma77_print_level** *(integer)*: Debug printing level for the linear solver MA77

(default = -1)

**ma77_small** *(real)*: Zero Pivot Threshold

Any pivot less than ma77_small is treated as zero.
(default = 1e-20)

**ma77_static** *(real)*: Static Pivoting Threshold

See MA77 documentation. Either ma77_static=0.0 or ma77_static>ma77_small. ma77_static=0.0 disables static pivoting.
(default = 0)

**ma77_u** *(real)*: Pivoting Threshold

See MA77 documentation.
Range: [0, 0.5]
(default = 1e-08)

**ma77_umax** *(real)*: Maximum Pivoting Threshold

Maximum value to which u will be increased to improve quality.
Range: [0, 0.5]
(default = 0.0001)

**ma86_nemin** *(integer)*: Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than ma86_nemin variables.
(default = 32)

**ma86_order** *(string)*: Controls type of ordering used by HSL_MA86

This option controls ordering for the solver HSL_MA86.
(default = auto)

  - amd Use the HSL_MC68 approximate minimum degree algorithm
  - auto Try both AMD and MeTiS, pick best
  - metis Use the MeTiS nested dissection algorithm (if available)

**ma86_print_level** *(integer)*: Debug printing level for the linear solver MA86

(default = -1)

**ma86_scaling** *(string)*: Controls scaling of matrix

This option controls scaling for the solver HSL_MA86.
(default = mc64)

  - mc64 Scale linear system matrix using MC64
  - mc77 Scale linear system matrix using MC77 [1,3,0]
none Do not scale the linear system matrix

**ma86_small** *(real)*: Zero Pivot Threshold

Any pivot less than `ma86_small` is treated as zero.
(default = $1e-20$)

**ma86_static** *(real)*: Static Pivoting Threshold

See MA86 documentation. Either `ma86_static=0.0` or `ma86_static>ma86_small`. `ma86_static=0.0` disables static pivoting.
(default = 0)

**ma86_u** *(real)*: Pivoting Threshold

See MA86 documentation.

Range: [0, 0.5]

(default = $1e-08$)

**ma86_umax** *(real)*: Maximum Pivoting Threshold

Maximum value to which `u` will be increased to improve quality.

Range: [0, 0.5]

(default = 0.0001)

**ma97_nemin** *(integer)*: Node Amalgamation parameter

Two nodes in elimination tree are merged if result has fewer than `ma97_nemin` variables.

(default = 8)

**ma97_order** *(string)*: Controls type of ordering used by HSL_MA97

(default = auto)

- amd Use the HSL_MC68 approximate minimum degree algorithm
- auto Use HSL_MA97 heuristic to guess best of AMD and METIS
- best Try both AMD and MeTiS, pick best
- matched-amd Use the HSL_MC80 matching based ordering with AMD
- matched-auto Use the HSL_MC80 matching with heuristic choice of AMD or METIS
- matched-metis Use the HSL_MC80 matching based ordering with METIS
- metis Use the MeTiS nested dissection algorithm

**ma97_print_level** *(integer)*: Debug printing level for the linear solver MA97

(default = 0)

**ma97_scaling** *(string)*: Specifies strategy for scaling in HSL_MA97 linear solver

(default = dynamic)

- dynamic Dynamically select scaling according to rules specified by `ma97_scalingX` and `ma97_switchX` options.
- mc30 Scale all linear system matrices using MC30
- mc64 Scale all linear system matrices using MC64
- mc77 Scale all linear system matrices using MC77 [1,3,0]
- none Do not scale the linear system matrix

**ma97_scaling1** *(string)*: First scaling.

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch1`. If `ma97_switch2` is triggered it is disabled.
(default = mc64)

mc30 Scale linear system matrix using MC30
mc64 Scale linear system matrix using MC64
mc77 Scale linear system matrix using MC77 [1,3,0]
none No scaling

**ma97.scaling2 (string):** Second scaling.

If ma97.scaling=dynamic, this scaling is used according to the trigger ma97.switch2. If ma97.switch3 is triggered it is disabled.

( default = mc64)

mc30 Scale linear system matrix using MC30
mc64 Scale linear system matrix using MC64
mc77 Scale linear system matrix using MC77 [1,3,0]
none No scaling

**ma97.scaling3 (string):** Third scaling.

If ma97.scaling=dynamic, this scaling is used according to the trigger ma97.switch3.

( default = mc64)

mc30 Scale linear system matrix using MC30
mc64 Scale linear system matrix using MC64
mc77 Scale linear system matrix using MC77 [1,3,0]
none No scaling

**ma97.small (real):** Zero Pivot Threshold

Any pivot less than ma97.small is treated as zero.

( default = 1e-20)

**ma97.solve.blas3 (string):** Controls if blas2 or blas3 routines are used for solve

( default = no)

no Use BLAS2 (faster, some implementations bit incompatible)
yes Use BLAS3 (slower)

**ma97.switch1 (string):** First switch, determine when ma97.scaling1 is enabled.

If ma97.scaling=dynamic, ma97.scaling1 is enabled according to this condition. If ma97.switch2 occurs this option is henceforth ignored.

( default = od_hd_reuse)

at_start Scaling to be used from the very start.
at_start_reuse Scaling to be used on first iteration, then reused thereafter.
high_delay Scaling to be used after more than 0.05*n delays are present
high_delay_reuse Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
never Scaling is never enabled.
od_hd Combination of on_demand and high_delay
od_hd_reuse Combination of on_demand_reuse and high_delay_reuse
on_demand Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse As on_demand, but reuse scaling from previous itr

**ma97_switch2 (string):** Second switch, determine when ma97_scaling2 is enabled.

If ma97_scaling=dynamic, ma97_scaling2 is enabled according to this condition. If ma97_switch3 occurs this option is henceforth ignored.

(default = never)

- **at_start** Scaling to be used from the very start.
- **at_start_reuse** Scaling to be used on first iteration, then reused thereafter.
- **high_delay** Scaling to be used after more than 0.05*n delays are present
- **high_delay_reuse** Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
- **never** Scaling is never enabled.
- **od_hd** Combination of on_demand and high_delay
- **od_hd_reuse** Combination of on_demand_reuse and high_delay_reuse
- **on_demand** Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
- **on_demand_reuse** As on_demand, but reuse scaling from previous itr

**ma97_switch3 (string):** Third switch, determine when ma97_scaling3 is enabled.

If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.

(default = never)

- **at_start** Scaling to be used from the very start.
- **at_start_reuse** Scaling to be used on first iteration, then reused thereafter.
- **high_delay** Scaling to be used after more than 0.05*n delays are present
- **high_delay_reuse** Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
- **never** Scaling is never enabled.
- **od_hd** Combination of on_demand and high_delay
- **od_hd_reuse** Combination of on_demand_reuse and high_delay_reuse
- **on_demand** Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
- **on_demand_reuse** As on_demand, but reuse scaling from previous itr

**ma97_u (real):** Pivoting Threshold

See MA97 documentation.

Range: [0, 0.5]

(default = 1e-08)

**ma97_umax (real):** Maximum Pivoting Threshold

See MA97 documentation.

Range: [0, 0.5]

(default = 0.0001)

**max_cpu_time (real):** Maximum number of CPU seconds.

A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding error message.
**max_filter_resets (integer):** Maximal allowed number of filter resets

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determines the maximal number of resets that are allowed to take place.

(default = 5)

**max_hessian_perturbation (real):** Maximum value of regularization parameter for handling negative curvature.

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. (This is delta_w^max in the implementation paper.)

(default = 1e+20)

**max_iter (integer):** Maximum number of iterations.

The algorithm terminates with an error message if the number of iterations exceeded this number.

(default = maxint)

**max_refinement_steps (integer):** Maximum number of iterative refinement steps per linear system solve.

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps.

(default = 10)

**max_resto_iter (integer):** Maximum number of successive iterations in restoration phase.

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number.

(default = 3000000)

**max_soc (integer):** Maximum number of second order correction trial steps at each iteration.

Choosing 0 disables the second order corrections. (This is p^(max) of Step A-5.9 of Algorithm A in the implementation paper.)

(default = 4)

**max_soft_resto_iters (integer):** Maximum number of iterations performed successively in soft restoration phase.

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called.

(default = 10)

**mehrotra_algorithm (string):** Indicates if we want to do Mehrotra’s algorithm.

If set to yes, Ipopt runs as Mehrotra’s predictor-corrector algorithm. This works usually very well for LPs and convex QPs. This automatically disables the line search, and chooses the (unglobalized) adaptive mu strategy with the “probing” oracle, and uses “corrector_type=affine” without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of ”bound_push”, ”bound_frac”, and ”bound_mult_init_val” are set more aggressive, and sets “alpha_for_y=bound_mult”.

(default = no)

no Do the usual Ipopt algorithm.

yes Do Mehrotra’s predictor-corrector algorithm.

**min_hessian_perturbation (real):** Smallest perturbation of the Hessian block.
The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. (This is delta_w^0min in implementation paper.)
(default = 1e-20)

**min_refinement_steps (integer):** Minimum number of iterative refinement steps per linear system solve.

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.)

(default = 1)

**mumps_dep_tol (real):** Pivot threshold for detection of linearly dependent constraints in MUMPS.

When MUMPS is used to determine linearly dependent constraints, this is determines the threshold for a pivot to be considered zero. This is CNTL(3) in MUMPS.

(default = 0)

**mumps_mem_percent (integer):** Percentage increase in the estimated working space for MUMPS.

In MUMPS when significant extra fill-in is caused by numerical pivoting, larger values of mumps_mem_percent may help use the workspace more efficiently. On the other hand, if memory requirement are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

(default = 1000)

**mumps_permuting_scaling (integer):** Controls permuting and scaling in MUMPS

This is ICNTL(6) in MUMPS.

Range: [0, 7]

(default = 7)

**mumps_pivot_order (integer):** Controls pivot order in MUMPS

This is ICNTL(7) in MUMPS.

Range: [0, 7]

(default = 7)

**mumps_pivot (real):** Pivot tolerance for the linear solver MUMPS.

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

(default = 1e-06)

**mumps_pivotmax (real):** Maximum pivot tolerance for the linear solver MUMPS.

Ipopt may increase pivot as high as pivotmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

(default = 0.1)

**mumps_scaling (integer):** Controls scaling in MUMPS

This is ICNTL(8) in MUMPS.

Range: [-2, 77]

(default = 77)

**mu_allow_fast_monotone_decrease (string):** Allow skipping of barrier problem if barrier test is already met.

If set to "no", the algorithm enforces at least one iteration per barrier problem, even if the barrier test is already met for the updated barrier parameter.
(default = yes)

- **no**: Take at least one iteration per barrier problem
- **yes**: Allow fast decrease of mu if barrier test it met

**mu_init (real):** Initial value for the barrier parameter.

This option determines the initial value for the barrier parameter (mu). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if “mu_strategy” is chosen as “monotone”)

(default = 0.1)

**mu_linear_decrease_factor (real):** Determines linear decrease rate of barrier parameter.

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of mu*”mu_linear_decrease_factor” and mu^”superlinear_decrease_power”. (This is kappa_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: [0, 1]

(default = 0.2)

**mu_max (real):** Maximum value for barrier parameter.

This option specifies an upper bound on the barrier parameter in the adaptive mu selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option ”mu_strategy” is chosen as ”adaptive”.)

(default = 100000)

**mu_max_fact (real):** Factor for initialization of maximum value for barrier parameter.

This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option ”mu_strategy” is chosen as ”adaptive”).

(default = 1000)

**mu_min (real):** Minimum value for barrier parameter.

This option specifies the lower bound on the barrier parameter in the adaptive mu selection mode. By default, it is set to the minimum of 1e-11 and min(”tol”;”compl_inf_tol”)/(”barrier_tol_factor”+1), which should be a reasonable value. (Only used if option ”mu_strategy” is chosen as ”adaptive”).

(default = 1e-11)

**mu_oracle (string):** Oracle for a new barrier parameter in the adaptive strategy.

Determines how a new barrier parameter is computed in each ”free-mode” iteration of the adaptive barrier parameter strategy. (Only considered if ”adaptive” is selected for option ”mu_strategy”).

(default = quality-function)

- **logo LOQO’s centrality rule**
- **probing Mehrotra’s probing heuristic**
- **quality-function minimize a quality function**

**mu_strategy (string):** Update strategy for barrier parameter.

Determines which barrier parameter update strategy is to be used.

(default = adaptive)

- **adaptive** use the adaptive update strategy
- **monotone** use the monotone (Fiacco-McCormick) strategy

**mu_superlinear_decrease_power (real):** Determines superlinear decrease rate of barrier parameter.

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of mu*”mu_linear_decrease_factor” and mu^”superlinear_decrease_power”. (This is theta_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.
Range: [1, 2]
(default = 1.5)

**mu_target** (real): Desired value of complementarity.

Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of “mu_target” specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

(default = 0)

**neg_curv_test_tol** (real): Tolerance for heuristic to ignore wrong inertia.

If positive, incorrect inertia in the augmented system is ignored, and we test if the direction is a direction of positive curvature. This tolerance determines when the direction is considered to be sufficiently positive.

(default = 0)

**nlp_scaling_constr_target_gradient** (real): Target value for constraint function gradient size.

If a positive number is chosen, the scaling factor the constraint functions is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the constraint functions.

(default = 0)

**nlp_scaling_max_gradient** (real): Maximum gradient after NLP scaling.

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is $g_{\text{max}}$ in Section 3.8 of the implementation paper.) Note: This option is only used if ”nlp_scaling_method” is chosen as ”gradient-based”.

(default = 100)

**nlp_scaling_method** (string): Select the technique used for scaling the NLP.

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP. If you are using AMPL, they can be specified through suffixes (”scaling_factor”)

(default = gradient-based)

  - equilibration-based scale the problem so that first derivatives are of order 1 at random points (only available with MC19)
  - gradient-based scale the problem so the maximum gradient at the starting point is scaling_max_gradient
  - none no problem scaling will be performed

**nlp_scaling_min_value** (real): Minimum value of gradient-based scaling values.

This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if ”nlp_scaling_method” is chosen as ”gradient-based”.

(default = 1e-08)

**nlp_scaling_obj_target_gradient** (real): Target value for objective function gradient size.

If a positive number is chosen, the scaling factor the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the objective function.

(default = 0)

**num_linear_variables** (integer): Number of linear variables
When the Hessian is approximated, it is assumed that the first num_linear_variables variables are linear. The Hessian is then not approximated in this space. If the get_number_of_nonlinear_variables method in the TNLP is implemented, this option is ignored.

\[(\text{default} = 0)\]

**nu_inc (real):** Increment of the penalty parameter.

\[(\text{default} = 0.0001)\]

**nu_init (real):** Initial value of the penalty parameter.

\[(\text{default} = 1e^{-06})\]

**obj_max_inc (real):** Determines the upper bound on the acceptable increase of barrier objective function.

Trial points are rejected if they lead to an increase in the barrier objective function by more than obj_max_inc orders of magnitude.

\[(\text{default} = 5)\]

**pardiso_matching_strategy (string):** Matching strategy to be used by Pardiso

This is IPAR(13) in Pardiso manual.

\[(\text{default} = \text{complete+2x2})\]

- complete Match complete (IPAR(13)=1)
- complete+2x2 Match complete+2x2 (IPAR(13)=2)
- constraints Match constraints (IPAR(13)=3)

**pardiso_max_iterative_refinement_steps (integer):** Limit on number of iterative refinement steps.

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

\[(\text{default} = 1)\]

**pardiso_msglvl (integer):** Pardiso message level

This determines the amount of analysis output from the Pardiso solver. This is MSGLVL in the Pardiso manual.

\[(\text{default} = 0)\]

**pardiso_order (string):** Controls the fill-in reduction ordering algorithm for the input matrix.

\[(\text{default} = \text{metis})\]

- amd minimum degree algorithm
- metis MeTiS nested dissection algorithm
- one undocumented
- pmetis parallel (OpenMP) version of MeTiS nested dissection algorithm

**pardiso_redo_symbolic_fact_only_if_inertia_wrong (string):** Toggle for handling case when elements were perturbed by Pardiso.

\[(\text{default} = \text{no})\]

- no Always redo symbolic factorization when elements were perturbed
- yes Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

**pardiso_repeated_perturbation_means_singular (string):** Interpretation of perturbed elements.

\[(\text{default} = \text{no})\]

- no Don’t assume that matrix is singular if elements were perturbed after recent symbolic factorization
yes Assume that matrix is singular if elements were perturbed after recent symbolic factorization

`pardiso_skip_inertia_check (string):` Always pretend inertia is correct.

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

(default = no)

no check inertia

yes skip inertia check

`perturb_always_cd (string):` Active permanent perturbation of constraint linearization.

This option makes the $\delta_c$ and $\delta_d$ perturbation be used for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.

(default = no)

no perturbation only used when required

yes always use perturbation

`perturb_dec_fact (real):` Decrease factor for $x-s$ perturbation.

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. (This is $\kappa_w^-$ in the implementation paper.)

Range: $[0, 1]$

(default = 0.333333)

`perturb_inc_fact (real):` Increase factor for $x-s$ perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. (This is $\kappa_w^+$ in the implementation paper.)

(default = 8)

`perturb_inc_fact_first (real):` Increase factor for $x-s$ perturbation for very first perturbation.

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for the first perturbation than that used for the remaining perturbations. (This is $\bar{\kappa}_w^+$ in the implementation paper.)

(default = 100)

`print_eval_error (string):` Switch to enable printing information about function evaluation errors into the GAMS listing file.

(default = yes)

no

yes

`print_frequency_iter (integer):` Determines at which iteration frequency the summarizing iteration output line should be printed.

Summarizing iteration output is printed every `print_frequency_iter` iterations, if at least `print_frequency_time` seconds have passed since last output.

(default = 1)

`print_frequency_time (real):` Determines at which time frequency the summarizing iteration output line should be printed.

Summarizing iteration output is printed if at least `print_frequency_time` seconds have passed since last output and the iteration number is a multiple of `print_frequency_iter`.

(default = 0)

`print_info_string (string):` Enables printing of additional info string at end of iteration output.

This string contains some insider information about the current iteration. For details, look for "Diagnostic Tags" in the Ipopt documentation.
The default verbosity level for console output. The larger this value the more detailed is the output.

Range: [0, 12]
(default = 5)

Switch to print timing statistics.
If selected, the program will print the CPU usage (user time) for selected tasks.

(default = no)

The balancing term included in the quality function for centrality.
This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. (Only used if option “mu_oracle” is set to “quality-function”.)

(default = none)

The penalty term for centrality that is included in quality function.
This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the xi in the Loqo update rule. (Only used if option “mu_oracle” is set to “quality-function”.)

(default = none)

Maximum number of search steps during direct search procedure determining the optimal centering parameter.
The golden section search is performed for the quality function based mu_oracle. (Only used if option “mu_oracle” is set to “quality-function”.)

(default = 8)

Norm used for components of the quality function.
(Only used if option “mu_oracle” is set to “quality-function”.)

(default = 2-norm-squared)

Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).
The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function").

Range: [0, 1]
(default = 0)

**quality_function_section_sigma_tol (real):** Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function").

Range: [0, 1]
(default = 0.01)

**recalc_y (string):** Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than recalc_y_feas_tol. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default.

(default = no)

- no use the Newton step to update the multipliers
- yes use least-square multiplier estimates

**recalc_y_feas_tol (real):** Feasibility threshold for recomputation of multipliers.

If recalc_y is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.

(default = 1e-06)

**replace_bounds (string):** Indicates if all variable bounds should be replaced by inequality constraints

This option must be set for the inexact algorithm

(default = no)

- no leave bounds on variables
- yes replace variable bounds by inequality constraints

**report_mininfeas_solution (string):** Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible.

This option allows to obtain the most feasible solution found by Ipopt during the iteration process, if it stops at a (locally) infeasible solution, due to a limit (time, iterations, ...), or with a failure in the restoration phase.

(default = no)

- no
- yes

**required_infeasibility_reduction (real):** Required reduction of infeasibility before leaving restoration phase.

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option.

Range: [0, 1]
(default = 0.9)

**residual_improvement_factor (real):** Minimal required reduction of residual test ratio in iterative refinement.

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted.

(default = 1)

**residual_ratio_max (real):** Iterative refinement tolerance
Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

(default = 1e-10)

residual_ratio_singular (real): Threshold for declaring linear system singular after failed iterative refinement.

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

(default = 1e-05)

resto_failure_feasibility_threshold (real): Threshold for primal infeasibility to declare failure of restoration phase.

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is 1e2*tol, where tol is the general termination tolerance.

(default = 0)

resto_penalty_parameter (real): Penalty parameter in the restoration phase objective function.

This is the parameter rho in equation (31a) in the Ipopt implementation paper.

(default = 1000)

resto_proximity_weight (real): Weighting factor for the proximity term in restoration phase objective.

This determines how the parameter zeta in equation (29a) in the implementation paper is computed. zeta here is resto_proximity_weight*sqrt(mu), where mu is the current barrier parameter.

(default = 1)

rho (real): Value in penalty parameter update formula.

Range: [0, 1]

(default = 0.1)

sigma_max (real): Maximum value of the centering parameter.

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

(default = 100)

sigma_min (real): Minimum value of the centering parameter.

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

(default = 1e-06)

skip_corr_if_neg_curv (string): Skip the corrector step in negative curvature iteration (unsupported!).

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive".

(default = yes)

no don’t skip

yes skip

skip_corr_in_monotone_mode (string): Skip the corrector step during monotone barrier parameter mode (unsupported!).

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive".

(default = yes)

no don’t skip

yes skip
slack_bound_frac \text{(real)}: Desired minimum relative distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: \([0, 0.5]\)

(default = 0.01)

slack_bound_push \text{(real)}: Desired minimum absolute distance from the initial slack to bound.

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

(default = 0.01)

slack_move \text{(real)}: Correction size for very small slacks.

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is mach\_eps \times 3/4. (See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.)

(default = 1.81899e-12)

soft_resto_perror_reduction_factor \text{(real)}: Required reduction in primal-dual error in the soft restoration phase.

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase.

(default = 0.9999)

start_with_resto \text{(string)}: Tells algorithm to switch to restoration phase in first iteration.

Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure.

(default = no)

no don’t force start in restoration phase

yes force start in restoration phase

s_max \text{(real)}: Scaling threshold for the NLP error.

(See paragraph after Eqn. (6) in the implementation paper.)

(default = 100)

s_phi \text{(real)}: Exponent for linear barrier function model in the switching rule.

(See Eqn. (19) in the implementation paper.)

(default = 2.3)

s_theta \text{(real)}: Exponent for current constraint violation in the switching rule.

(See Eqn. (19) in the implementation paper.)

(default = 1.1)

tau_min \text{(real)}: Lower bound on fraction-to-the-boundary parameter tau.

(This is tau_min in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: \([0, 1]\)

(default = 0.99)

theta_max_fact \text{(real)}: Determines upper bound for constraint violation in the filter.

The algorithmic parameter theta_max is determined as theta_max_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta_max is unacceptable to the filter.
theta_min_fact (real): Determines constraint violation threshold in the switching rule.

The algorithmic parameter theta_min is determined as theta_min_fact times the maximum of 1 and the constraint violation at initial point. The switching rules treat an iteration as an h-type iteration whenever the current constraint violation is larger than theta_min (see paragraph before Eqn. (19) in the implementation paper).

(default = 0.0001)

tiny_step_tol (real): Tolerance for detecting numerically insignificant steps.

If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.

(default = 2.22045e-15)

tiny_step_y_tol (real): Tolerance for quitting because of numerically insignificant steps.

If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny_step_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.

(default = 0.01)

tol (real): Desired convergence tolerance (relative).

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "constr_viol_tol", and "compl_inf_tol" are met. (This is epsilon_tol in Eqn. (6) in implementation paper). See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.

(default = 1e-08)

warm_start_bound_frac (real): same as bound_frac for the regular initializer.

Range: [0, 0.5]

(default = 0.001)

warm_start_bound_push (real): same as bound_push for the regular initializer.

(default = 0.001)

warm_start_init_point (string): Warm-start for initial point

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)

(default = no)

no do not use the warm start initialization

yes use the warm start initialization

warm_start_mult_bound_push (real): same as mult_bound_push for the regular initializer.

(default = 0.001)

warm_start_mult_init_max (real): Maximum initial value for the equality multipliers.

(default = 1e+06)

warm_start_slack_bound_frac (real): same as slack_bound_frac for the regular initializer.

Range: [0, 0.5]

(default = 0.001)

warm_start_slack_bound_push (real): same as slack_bound_push for the regular initializer.

(default = 0.001)
**watchdog_shortened_iter_trigger (integer):** Number of shortened iterations that trigger the watchdog.

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing “0” here disables the watchdog procedure.

(default = 10)

**watchdog_trial_iter_max (integer):** Maximum number of watchdog iterations.

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.

(default = 3)

### Bibliography


JAMS - an EMP solver

1 Introduction

EMP (Extended Mathematical Programming) is not a solver but an (experimental) framework for automated mathematical programming reformulations. The idea behind EMP is that new upcoming types of models which currently cannot be solved reliably are reformulated into models of established math programming classes in order to use mature solver technology. At this stage, EMP supports the modeling of Bilevel Programs, Variational Inequalities, Disjunctive Programs, Extended Nonlinear Programs and Embedded Complementarity Systems.

Extended mathematical programs are collections of functions and variables joined together using specific optimization and complementarity primitives. EMP annotates the existing relationships within a model to facilitate higher level structure identification. A specific implementation of this framework is outlined that reformulates the original GAMS model automatically using directives contained in an "empinfo" file into an equivalent model that can be solved using existing GAMS solvers.

The reformulation is done by the solver JAMS which currently is the only solver that is capable of handling EMP models. Examples showing how to use the EMP framework and the solver JAMS are made available through the GAMS EMP Library which is included in the GAMS Distribution. In order to generate a copy of an EMPLIB model, one can use the library facility of the GAMS IDE, or execute the command line directive emplib <modelname>, where modelname is the (stem of the) file containing the model.
EMP has been developed jointly by Michael Ferris of UW-Madison, Ignacio Grossmann of Carnegie Mellon University, and GAMS Development Corporation. EMP and JAMS come free of charge with any licensed GAMS system but require a subsolver to solve the generated models.

2 JAMS: a reformulation tool

EMP models are currently processed by the JAMS solver. The solver JAMS creates a scalar version of the given GAMS model. This scalar version of the model is then solved by an appropriate subsolver. By default, there are no reformulations carried out, so the model generated is simply a GAMS scalar form of the model the actual subsolver will process. The subsolver used is by default the currently specified solver for the given model type.

2.1 The JAMS Option File

As with any GAMS solver, JAMS has an option file, typically called jams.opt. A JAMS option subsolver is available to change the subsolver used for the reformulated model, along with an option to utilize a subsolver option file (subsolveropt). The actual scalar version of the model can also be seen by the modeler using the option filename. For example, the option file

```plaintext
subsolver path
subsolveropt 1
filename mcpmod.gms
```

when applied to an EMP model that is a complementarity problem will create a file called mcpmod.gms in the current directory and solve that model using the solver PATH utilizing any options for PATH that are specified in path.opt. The scalarized model is not particularly useful to look at since all of the original variables have been renamed into a scalar form. The mapping between original variables and the ones used in the scalar version of the model is given in a dictionary file that can also be seen by the modeler using the dict option. If a user simply wants to generate this scalar model, then the option terminate will not solve the generated model.

After the scalar version of the model is solved, the solution values are mapped back into the original namespace and returned to the modeler as usual in the listing file. The JAMS option margtol allows the modeler to suppress reporting marginals that have (absolute) values smaller than this tolerance.

Obviously, all of the above functionality is not of much value: the key part of JAMS is to interpret additional directives to take the original model and produce a reformulated scalar model. This is carried out using an "empinfo" file. The syntax and use of this file is the content of the remaining sections of this document.

The option EMPInfoFile allows the user to specify the path and name of a file containing additional EMP information. The syntax of this file is described by examples elsewhere in this document and in Section Empinfo file details. There is a subtlety that should be mentioned when a user writes this file in the default location in the GAMS scratch directory of the current run specified in the gams file using:

```plaintext
file empinfo / '%emp.info%' /;
```

wherein certain additional formatting instructions are given. If instead, the file handle is given by:

```plaintext
file empinfo / 'empinfo.txt' /;
```

for example, then to produce the same file the following lines must be added to the gams source file:

```plaintext
empinfo.pc = 8;
empinfo.pw = 255;
```

The following tables list all available options.

2.2 Reformulation Options
### 2.3 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeepObj</td>
<td>Keep original objective function and variables in generated MCP model. This option is only valid for reformulations into complementarity systems. If KeepObj is set the generated MCP program will incorporate the original objective function and the Karush-Kuhn-Tucker condition of the original objective variable.</td>
<td>0</td>
</tr>
<tr>
<td>MargTol</td>
<td>Only report marginals with an absolute value above the tolerance</td>
<td>1e-6</td>
</tr>
<tr>
<td>ObjVarName</td>
<td>Name of objective variable in generated model.</td>
<td>objvar</td>
</tr>
</tbody>
</table>

### 3 Forming Optimality Conditions: NLP2MCP

The first nontrivial use of the JAMS solver is to automatically generate the first order conditions of a linear or nonlinear program; essentially we reformulate the optimization problem as a mixed complementarity problem (MCP). The “empinfo” file to do this simply contains the following line:

```plaintext
modeltype mcp
```

Behind the scenes, JAMS forms the Lagrangian of the nonlinear program and then forms its Karush-Kuhn-Tucker optimality conditions. To be clear, given the original nonlinear program

\[
\min_x f(x) \text{ s.t. } g(x) \leq 0, \quad h(x) = 0,
\]  

(NLP)

the Lagrangian is:

\[
\mathcal{L}(x, \lambda, \mu) = f(x) - \langle \lambda, g(x) \rangle - \langle \mu, h(x) \rangle.
\]

The first order conditions are the following MCP:

\[
0 = \nabla_x \mathcal{L}(x, \lambda, \mu) \quad \perp x \text{ free}
\]

\[
0 \geq -\nabla_\lambda \mathcal{L}(x, \lambda, \mu) \quad \perp \lambda \leq 0
\]

\[
0 = -\nabla_\mu \mathcal{L}(x, \lambda, \mu) \quad \perp \mu \text{ free}
\]
A specific example is:

\[
\begin{align*}
\min_{x,y,z} & \quad -3x + y \\
\text{s.t.} & \quad x + y \leq 1, \ x + y - z = 2, \ x, y \geq 0
\end{align*}
\]

which is found in the EMPLIB model \texttt{nlp2mcp}:

1 Positive Variables \(x, y\);  
2 Variables \(f, z\);  
3 Equations \(g, h, \text{defobj}\);  
4  
5 \(g.. \quad x + y \quad =l= \quad 1;\)  
6 \(h.. \quad x + y - z \quad =e= \quad 2;\)  
7 \(\text{defobj}. \quad f \quad =e= \quad -3x + y;\)  
8  
9 model \text{comp} / \text{defobj, g, h} /;  
10  
11 file \text{info} / \%\text{emp.info}\% /;  
12 putclose \text{info} / \text{modeltype mcp};  
13  
14 solve \text{comp} using emp minimizing \(f\);  

Lines 11-12 write out the default ”empinfo” file whose location is provided in the system string \%\text{emp.info}\%. Armed with this additional information, the EMP tool automatically creates the following MCP:

\[
\begin{align*}
0 \leq -3 - \lambda - \mu & \quad \perp \quad x \geq 0 \\
0 \leq 1 - \lambda - \mu & \quad \perp \quad y \geq 0 \\
0 & = \mu \quad \perp \quad z \text{ free} \\
0 \geq x + y - 1 & \quad \perp \quad \lambda \leq 0 \\
0 = x + y - z - 2 & \quad \perp \quad \mu \text{ free}.
\end{align*}
\]

### 4 Soft Constraints

In many cases, we wish to relax certain constraints in a model during solution (to help identify feasibility issues for example). As an example, consider the problem

\[
\begin{align*}
\min_{x_1, x_2, x_3} & \quad \exp(x_1) \\
\text{s.t.} & \quad \log(x_1) = 1, \\
& \quad x_1^2 \leq 2, \\
& \quad x_1/x_2 = \log(x_3), \\
& \quad 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0,
\end{align*}
\]

which can be formulated in GAMS as

1 $title simple example of ENLP  
2  
3 Variables obj, x1, x2, x3;  
4 Equations f0, f1, f2, f3, f4;  
5  
6 f0.. obj =e= \exp(x1);  
7 f1.. \log(x1) =e= 1;  
8 f2.. x1^2 =g= 2;  
9 f3.. x1/x2 =e= \log(x3);  
10 f4.. 3*x1 + x2 =l= 5;  
11  
12 x1.lo = 0; x2.lo = 0;  
13  
14 model enlpemp /all/;  
15  
16 solve enlpemp using nlp min obj;
4.1 Reformulation as a classical NLP

Soft constraints allow us to treat certain equations in the model as "soft" by removing the constraints and adding a penalty term to the objective function. Explicitly, we replace the above problem by:

\[
\begin{align*}
\min_{x_1, x_2, x_3} & \quad \exp(x_1) + 5 \| \log(x_1) - 1 \|^2 + 2 \max(x_2^2 - 2, 0) \\
\text{s.t.} & \quad x_1/x_2 = \log(x_3), \\
& \quad x_1 + x_2 \leq 5, x_1, x_2 \geq 0.
\end{align*}
\]

In this problem, we still force \(x_1/x_2 = \log(x_3)\), but apply a least squares penalty to \(\log(x_1) - 1\) and a smaller one-sided penalization to \(x_2^2 - 2\).

The above formulation is nonsmooth due to the "max" term in the objective function; in practice we would replace this by:

\[
\begin{align*}
\min_{x_1, x_2, x_3, w} & \quad \exp(x_1) + 5 (\log(x_1) - 1)^2 + 2w \\
\text{s.t.} & \quad x_1/x_2 = \log(x_3), \\
& \quad 3x_1 + x_2 \leq 5, x_1, x_2 \geq 0 \\
& \quad w \geq x_2^2 - 2, w \geq 0
\end{align*}
\]

and recover a standard form NLP.

The "empinfo" file:

```plaintext
modeltype NLP
adjustequ
f1 sqr 5
f2 maxz 2
```

coupled with replacing line 16 with

```plaintext
solve enlpemp using emp min obj;
```

achieves this goal. The parameter values provide the penalty coefficients above.

4.2 Reformulation as an MCP

As an alternative, we can rewrite the problem as an MCP, also dealing explicitly with the nonsmoothness. The "empinfo" file is given by:

```plaintext
modeltype NLP
adjustequ
f1 sqr 5
f2 maxz 2
```

and this generates the following MCP:

\[
\begin{align*}
0 &= \log(x_1) - 1 + y_1/10 \quad \perp y_1 \text{ free}, \\
0 &\leq x_2^2 - 2 \quad \perp y_2 \geq 0, \\
0 &= x_1/x_2 - \log(x_3) \quad \perp y_3 \text{ free}, \\
0 &\geq 3x_1 + x_2 - 5 \quad \perp y_4 \leq 0, \\
0 &\leq \exp(x_1) - y_1/x_1 - y_3/x_2 - 3y_4 \quad \perp x_1 \geq 0, \\
0 &\leq -2y_2x_2 + x_1y_3/x_2^2 - y_4 \quad \perp x_2 \geq 0, \\
0 &= y_3/x_3 \quad \perp x_3 \text{ free},
\end{align*}
\]

where \(y\) represent the multipliers.

A complete description of the process to derive this MCP will be given later in Section Extended Nonlinear Programs.
5 Bilevel Programs

Mathematical programs with optimization problems in their constraints have a long history in operations research including [2, 3, 5]. New codes are being developed that exploit this structure, at least for simple hierarchies, and attempt to define and implement algorithms for their solution.

The simplest case is that of bilevel programming, where an upper level problem depends on the solution of a lower level optimization. For example:

\[
\begin{align*}
\min_{x,y} f(x,y) \\
\text{s.t. } g(x,y) &\leq 0, \\
y \text{ solves } \min_{y} v(x,y) \text{ s.t. } h(x,y) &\geq 0.
\end{align*}
\]

Often, the upper level is referred to as the "leader", while the lower level is the "follower".

This problem can be reformulated as a Mathematical Program with Complementarity Constraints (MPCC) by replacing the lower level optimization problem by its first order optimality conditions:

\[
\begin{align*}
\min_{x,y} f(x,y) \\
\text{s.t. } g(x,y) &\leq 0, \\
0 = \nabla_y v(x,y) - \lambda^T \nabla_y h(x,y) &\perp x \text{ free} \\
0 &\leq h(x,y) \perp \lambda \geq 0.
\end{align*}
\]

We find a solution of the MPCC, not of the bilevel program. This approach allows the MPCC to be solved using the NLPEC code, for example. Note that this reformulation is potentially problematic. First order conditions require theoretical assumptions to be necessary and sufficient for local optimality. There may be cases where the lower level problem has multiple local solutions, but the modeler really was interested in the global solution. The approach here may not produce this solution, even if a global solver is used within NLPEC.

The following example is Example 5.1.1, page 197 from [2]. Mathematically, the problem is

\[
\begin{align*}
\min_{x,y} x - 4y \\
\text{s.t. } y \text{ solves } \min_{y} y \\
\text{s.t. } x + y &\geq 3 \\
2x - y &\geq 0 \\
-2x - y &\geq -12 \\
-3x + 2y &\geq -4
\end{align*}
\]

and the EMPLIB model bard511 contains the following code:

```
1  Positive variables x, y;
2  Variables objout, objin;
3  equations defout, defin, e1, e2, e3, e4;
4  5  defout.. objout =e= x - 4*y;
6  defin.. objin =e= y;
7  8 e1.. x + y =g= 3;
9  e2.. 2*x - y =g= 0;
10 e3.. -2*x - y =g= -12;
11 e4.. -3*x + 2*y =g= -4;
12 13 model bard / all /;
14 15 $echo bilevel x min objin y defin e1 e2 e3 e4 > "emp.info"
16 17 solve bard using emp minimizing objout;
```
Note that lines 1-13 define the functions that form the objectives and constraints of the model and assemble them into the model. Line 15 writes the "empinfo" file and states that the lower level problem involves the objective \( \text{objin} \) which is to be minimized by choice of variables \( y \) subject to the constraints specified in (defin), e1, e2, e3, and e4.

Note that the variables \( x \) are declared to be variables of the upper level problem and this example has no upper level constraints \( g \). Having written the problem in this way, the MPCC is generated automatically, and passed on to a solver. In the case where that solver is NLPEC, a further reformulation of the model is carried out to convert the MPCC into an equivalent NLP or a parametric sequence of NLP's.

Further examples of bilevel models in EMPLIB are named: bard*, ccmg74, ccmg153, flds*, jointc1, jointc2, mirrlees, transbp.

The EMP model type allows multiple lower level problems to be specified within the bilevel format. An example of this is given in EMPLIB as ccmg71. The equations and objectives are specified in the normal manner; the only change is the definition of the "empinfo" file, shown below as lines 8-12:

```plaintext
1 ... 2
3 defh1.. h1 =e= sqr(u1-x1) + sqr(u2-x2) + sqr(u3-x3) + sqr(u4-x4); 4 e1.. 3*u1 + u2 + 2*u3 + u4 =e= 6; 5
6 ... 7
8 $onecho > "emp.info%" 9 bilevel x1 x2 x3 x4 10 min h1 u1 u2 u3 u4 defh1 e1 11 min h2 v1 v2 v3 v4 defh2 e2 12 $offecho
```

This corresponds to a bilevel program with two followers, both solving minimization problems. The first follower minimizes the objective function \( h1 \) (defined in defh1 on line 3) over the variables u1, u2, u3, and u4 subject to the constraint given in e1. The second followers problem is defined analogously on line 11. Note that \( h1 \) involves the variables \( x1, x2, x3, \) and \( x4 \) that are optimization variables of the leader. The constraint in e1 could also include these variables, and also the variables \( v1, v2, v3, \) or \( v4 \) of the second follower, but all of these would be treated as parameters by the first follower.

The actual model (ccmg71) in EMPLIB uses a shortcut notation to replace lines 8-12 above by:

```plaintext
1 ... 2
3 defh1.. h1 =e= sqr(u1-x1) + sqr(u2-x2) + sqr(u3-x3) + sqr(u4-x4); 4 e1.. 3*u1 + u2 + 2*u3 + u4 =e= 6; 5
6 ... 7
8 $onecho > "emp.info%" 9 bilevel x1 x2 x3 x4 10 min h1 * defh1 e1 11 min h2 * defh2 e2 12 $offecho
```

In the followers problem defined on line 10, the '*' notation indicates that this agent will optimize over all the variables used in defh1 and e1 that are not under the control of any other follower or the leader. In this case, this means u1, u2, u3, and u4. To avoid confusion, it is recommended that the modeler explicitly names all the variables in each followers problem as shown before.

### 6 Variational Inequalities

A variational inequality \( VI(F,X) \) is to find \( x \in X \):

\[
\langle F(x), (z-x) \rangle \geq 0, \text{ for all } z \in X.
\]

Here \( X \) is a closed (frequently assumed convex) set, defined for example as

\[
X = \{ x \mid x \geq 0, h(x) \geq 0 \}.
\]
Note that the first-order (minimum principle) conditions of a nonlinear program

$$\min_{z \in X} f(z)$$

are precisely of this form with $F(x) = \nabla f(x)$.

It is well known that such problems can be reformulated as complementarity problems when the set $X$ has the representation

$$\{ x \mid x \geq 0, h(x) \geq 0 \}$$

by introducing multipliers $\lambda$ on the constraints $h(x) \geq 0$:

$$0 \leq F(x) - \langle \lambda, \nabla h(x) \rangle \perp x \geq 0$$

$$0 \leq h(x) \perp \lambda \geq 0.$$

If $X$ has a different representation, this construction would be modified appropriately.

A simple two dimensional example may be useful to improve understanding. Let

$$F(x) = \begin{bmatrix} x_1 + 2 \\ x_1 + x_2 - 3 \end{bmatrix}, X = \{ x \geq 0 \} : x_1 + x_2 \leq 1,$$

so that $F$ is an affine function, but $F$ is not the gradient of any function $f : \mathbb{R}^2 \to \mathbb{R}$. For this particular data, $VI(F, X)$ has a unique solution $x = (0, 1)$.

1 Set $J / 1, 2 /$;
2 Positive Variable $x(J) \ 'vars, perp to f(J)';$
3 4 Equations $F(J), h;$
5 6 $F(J) .. (x('1') + 2) \text{sameas}(J,'1') + (x('1') + x('2') - 3) \text{sameas}(J,'2') =n= 0 ;$
7 $h .. x('1') + x('2') =l= 1;$
8 9 model simpleVI / $F, h/;$
10 11 file fx /"%emp.info%"/;
12 putclose fx 'vi F x h';
13 14 solve simpleVI using emp;

Note that lines 1-9 of this file define the $F$ and $h$ using standard GAMS syntax and include the defining equations in the model simpleVI. The extension is the annotation "empinfo" file that indicates certain equations are to be treated differently by the EMP tool. The annotation simply says that the model is a VI (vi) that pairs $F$ with $x$ and that the remaining (unpaired) equations form the constraint set $X$. (Alternative notation allows the keyword vi to be replaced by vifunc.) Thus, model equations $F$ define a function $F$ that is to be part of a variational inequality, while the equations $h$ define constraints of $X$. It is also acceptable in this setting to use the "empinfo" file defined by:

putclose fx 'vi F x';

In this case, by default, any equations that are given in the model statement but not included as a pair in the vi statement are automatically used to form $X$. An alternative way to write this model without using sameas is given in EMPLIB as affinevi.

Some subtleties related to VI’s are demonstrated in the EMPLIB model: zerofunc. In this model, the set $X$ is defined using variables $y$ and $z$, for which $z$ does not appear in the definition of $F \equiv F_y$. In this case, the variable $z$ is then matched with a "0" function. The "empinfo" file can be written in a number of different ways:

putclose fx 'vi F_y y';

or

putclose fx 'vi z F_y y';

or

putclose fx 'vi z F_y y gCons';
or

\texttt{putclose fx 'vi \textit{F_z} z F_y y gCons'};

where \textit{F_z} simply defines the zero function. To some extent, our preferred notation is the one listed third: it explicitly includes all the variables and constraints that are present in the model and does not require the modeler to define \textit{F_z} at all.

Further example models in EMPLIB are \texttt{simplevi, simplevi2, simplevi3, target, traffic, traffic2, and transvi.}

Note also that the lower level problems of a bilevel program could be VI's instead of optimization problems - these problems are called Mathematical Programs with Equilibrium Constraints (MPEC) in the literature. Note that since MPCC is a special case of MPEC, the GAMS model type MPEC covers both. An example demonstrating this setup is given in EMPLIB as \texttt{multmpec.} The actual model to solve is:

\[
\min z \\
\text{s.t. } \exp(z) + y = 2, z \geq 1 \\
\quad (u,x) \text{ solves VI}([x+y+z-1;u-\log(x)];\{(u,x) \mid u \geq 0, x \geq 0\}) \\
\quad y \text{ solves VI}(y+z+3,\{y \mid y \text{ free}\})
\]

Note that the two VI's (due to the definitional sets) correspond respectively to a complementarity problem:

\[
0 \leq x+y+z-1 \perp u \geq 0 \\
0 \leq u-\log(x) \perp x \geq 0
\]

and a linear equation:

\[
y + z + 3 = 0
\]

The actual GAMS code is as follows:

```
1 Variables x, y, z;
2   Positive Variable u;
3 Equations f, f2, g, h;
4
5 f.. x + y + z =e= 1;
6 f2.. u =e= log(x);
7
8 g.. y + z =n= -3;
9
10 h.. exp(z) + y =n= 2;
11
12 x.lo = 0; x.l = 1; z.lo = 1;
13
14 model mpecmod /all/;
15
16 $onecho > %emp.info%
17 bilevel
18 vi u f
19 x f2
20 vi g y
21 $offecho
22
23 solve mpecmod using emp min z;
```

A starting value for \textit{x} is needed to protect the evaluation of \textit{\log(x)} on line 12, while the two complementarity problems are given on lines 18-19, and line 20 respectively (it is not really necessary to split these VI's apart, but it may give information to a solver that can be exploited). It is of course possible to write the MPCC directly in GAMS using the model type MPEC instead of EMP.
7 Embedded Complementarity Systems

A different type of embedded optimization model that arises frequently in applications is:

$$\max_x f(x,y)$$
\[ \text{s.t. } g(x,y) \leq 0 \quad (\bot p \geq 0) \]
\[ H(x,y,p) = 0 \quad (\bot y \text{ free}) \]

Note the difference here: the optimization problem is over the variable $x$, and is parameterized by the variable $y$. The choice of $y$ is fixed by the (auxiliary) complementarity relationships depicted here by $H$. Note that the $H$ equations are not part of the optimization problem, but are essentially auxiliary constraints to tie down remaining variables in the model.

A specific example is:

$$\max_x x$$
\[ \text{s.t. } x + y \leq 1 \]
\[ -3x + y = 0.5 \quad (\bot y \text{ free}) \]

which is found in the EMPLIB model simpequil2:

```
1 Variables y;
2 Positive variables x;
3 Equations optcons, vicons;
4 5 optcons.. x + y =l= 1;
6 vicons.. -3*x + y =e= 0.5;
7 model comp / optcons, vicons /;
8 file info / '%emp.info%' /;
9 put info / 'equilibrium';
10 put / 'max x optcons';
11 putclose / 'vi vicons y';
12 solve comp using emp;
```

In order that this model can be processed correctly as an EMP, the modeler provides additional annotations to the model defining equations (lines 1-8 above) in an "empinfo" file (lines 10-13). Specifically, line 11 indicates the problem is an equilibrium problem involving one or more agent problems. Line 12 defines the first agent as an optimizer (over $x$), and line 13 defines the second agent as solving a VI in $y$. Armed with this additional information, the EMP tool automatically creates the following MCP:

\[
\begin{align*}
0 & \leq -1 + p & \bot x \geq 0 \\
0 & \leq 1 - x - y & \bot p \geq 0 \\
0 & = -3x + y - 0.5 & \bot y \text{ free},
\end{align*}
\]

(which is formed by the steps we outline below). EMP explicitly enforces the rule that every variable and constraint is under the control of exactly one agent. Thus a constraint or a variable cannot appear in both the max problem and the VI problem.

The above example is slightly simpler than the general form described above in which $H$ is a function of $x$, $y$, and $p$, the multiplier on the constraint of the optimization problem. The problem is that we do not have that variable around in the model code if we only specify the optimization problem there. This occurs for example in the classical PIES Model due to Hogan. In this setting, the problem is described by a linear program

\[
\begin{align*}
\min_x & \quad c^T x \\
\text{s.t. } & \quad Ax = q(p) \\
& \quad Bx = b \\
& \quad x \geq 0
\end{align*}
\]

in which the quantity $q$ is a function of $p$, which is a multiplier on one of the LP constraints. To do this in EMP, we simply add the annotation:

```
1 model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
2```
where `dembal` is the name of the constraint for which `p` needs to be the multiplier. The full model is found in the EMPLIB model `pies`. Two final points: the `dualvar` directive identifies the variable `p` with the multiplier on the `dembal` constraint, and all variables and constraints must be owned by a single agent. In this case, since there is only one agent (the minimizer), all constraints of the model are explicitly claimed in line 5, along with all variables except for `p`. However, line 5 identifies `p` with the `dembal` constraint which is owned by the min agent, and hence `p` is also owned by that agent.

There are several shorthands possible here. The first is that line 5 can be replaced by the `∗` form:

```plaintext
1 model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
2
3 file myinfo /'emp.info%'/;
4 put myinfo 'equilibrium ';
5 put 'min obj * defobj dembal cmbal ombal lmbal hmbal ruse ';
6
7 solve piesemp using emp;
```

Alternatively, an even shorter version is possible since there is only one agent present in this model, namely:

```plaintext
1 model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /
2
3 file myinfo /'emp.info%'/;
4 putmyinfo 'dualvar p dembal';
5
6 solve piesemp using emp minimizing obj;
```

Note that in this form, all the variables and constraints of the original model are included in the (single) agents problem, and the original variable `p` is identified in the constructed MCP with the multiplier on the `dembal` constraint.

In the general case where the "empinfo" file contains all three lines:

```plaintext
min x optcons
vi viscons y
dualvar p optcons
```

namely that the function `H` that is defined in `viscons` is complementary to the variable `y` (and hence the variable `y` is a parameter to the optimization problem), and furthermore that the dual variable associated with the equation `optcons` in the optimization problem is one and the same as the variable `p` used to define `H`, the EMP tool automatically creates the following MCP:

\[
\begin{align*}
0 &= \nabla_x \mathcal{L}(x,y,p) \quad \perp x \text{ free} \\
0 &\geq -\nabla_p \mathcal{L}(x,y,p) \quad \perp p \leq 0 \\
0 &= H(x,y,p) \quad \perp y \text{ free},
\end{align*}
\]

where the Lagrangian is defined as

\[
\mathcal{L}(x,y,p) = f(x,y) - \langle p, g(x,y) \rangle.
\]

Essentially, this MCP consists of the first order optimality conditions of the optimization problem, coupled with the VI that is the second agents problem. An example that does both of these things together is provided in EMPLIB as `scarfemp-primal`.

Note that since the PIES model has no `y` variables, this is a special case of the general form in which the second agents (VI) problem is simply not present.

Example models are `ferris43`, `flipper`, `pies`, `scarfemp-dual`, `simpequil`, `transecs`, and `transeql`.

## 8 MOPECs

MOPECs (Multiple Optimization Problems with Equilibrium Constraints) are a simple conceptual extension of the aforementioned embedded complementarity system. Instead of having a single optimizing agent and a variational inequality, one instead allows multiple optimizing agents coupled with an equilibrium constraint (the variational inequality).
Perhaps the most popular use of this formulation is where competition is allowed between agents. A standard method to deal with such cases is via the concept of Nash Games. In this setting $x^*$ is a Nash Equilibrium if

$$x^*_i \in \arg \min_{x_i \in X_i} f_i(x_i, x^*_{-i}, q), \forall i \in \mathcal{I},$$

where $x_{-i}$ are other players decisions and the quantities $q$ are given exogenously, or via complementarity:

$$0 \leq H(x, q) \perp q \geq 0.$$

This mechanism is extremely popular in economics, and Nash famously won the Nobel Prize for his contributions to this literature.

This format is again an EMP, more general than the example given above in two respects. Firstly, there is more than one optimization problem specified in the embedded complementarity system. Secondly, the parameters in each optimization problem consist of two types. Firstly, there are the variables $q$ that are tied down by the auxiliary complementarity condition and hence are treated as parameters by the $i$th Nash player. Also there are the variables $x_{-i}$ that are treated as parameters by the $i$th Nash player, but are treated as variables by a different player $j$.

While we do not specify the syntax here for these issues, EMPLIB provides examples that outline how to carry out this matching within GAMS. Finally, two points of note: first it is clear that the resulting model is a complementarity problem and can be solved using PATH, for example. Secondly, performing the conversion from an embedded complementarity system or a Nash Game automatically is a critical step in making such models practically useful.

We note that there is a large literature on discrete-time finite-state stochastic games: this has become a central tool in analysis of strategic interactions among forward-looking players in dynamic environments. The model of dynamic competition in an oligopolistic industry given in [4] is exactly in the format described above, and has been used extensively in applications such as advertising, collusion, mergers, technology adoption, international trade and finance. Ongoing work aims to use the EMP format to model these problems.

### 9 Extended Nonlinear Programs

Optimization models have traditionally been of the form (NLP). Specialized codes have allowed certain problem structures to be exploited algorithmically, for example simple bounds on variables. However, for the most part, assumptions of smoothness of $f$, $g$ and $h$ are required for many solvers to process these problems effectively. In a series of papers, Rockafellar and colleagues [8–10] have introduced the notion of extended nonlinear programming, where the (primal) problem has the form:

$$\min_{x \in X} f(x) + \theta(g_1(x), \ldots, g_m(x)). \quad \text{(ENLP)}$$

In this setting, $X$ is assumed to be a nonempty polyhedral set, and the functions $f, g_1, \ldots, g_m$ are smooth. The function $\theta$ can be thought of as a generalized penalty function that may well be nonsmooth. However, when $\theta$ has the form

$$\theta(u) = \sup_{y \in Y} \{ \langle y, u \rangle - k(y) \}, \quad \text{($\theta$ conj)}$$

a computationally exploitable and theoretically powerful framework can be developed based on conjugate duality. A key point for computation and modeling is that the function $\theta$ can be fully described by defining the set $Y$ and the function $k$. Furthermore, from a modeling perspective, an extended nonlinear program can be specified simply by defining the functions $f, g_1, \ldots, g_m$ in the manner already provided by the modeling system, with the additional issue of simply defining $Y$ and $k$. Conceptually, this is not much harder than what is carried out already, but leads to significant enhancements to the types of models that are available. Once a modeler determines which constraints are treated via which choice of $k$ and $Y$, the EMP model interface automatically forms an equivalent variational inequality or complementarity problem. As we show later, there may be alternative formulations that are computationally more appealing; such reformulations can be generated using different options to JAMS.

#### 9.1 Forms of theta

The EMP model type makes the problem format (ENLP) available to users in GAMS. As special cases, we can model piecewise linear penalties, least squares, and $L_1$ approximation problems, as well as the notion of soft and hard constraints.
For ease of exposition, we now describe a subset of the types of functions $\theta$ that can be generated by particular choices of $Y$ and $k$. In many cases, the function $\theta$ is separable, that is

$$\theta(u) = \sum_{i=1}^{m} \theta_i(u_i),$$

so we can either specify $\theta_i$ or $\theta$ itself.

Extended nonlinear programs include the classical nonlinear programming form (NLP) as a special case. This follows from the observation that if $K$ is a closed convex cone, and we let $\psi_K$ denote the "indicator function" of $K$ defined by:

$$\psi_K(u) = \begin{cases} 0 & \text{if } u \in K, \\ \infty & \text{else}, \end{cases}$$

then (NLP) can be rewritten as:

$$\min_x f(x) + \psi_K((g(x), h(x))), \quad K = \mathbb{R}^m \times \{0\}^p,$$

where $m$ and $p$ are the dimensions of $g$ and $h$ respectively and $\mathbb{R}^m = \{u \in \mathbb{R}^m \mid u \leq 0\}$. An elementary calculation shows that

$$\psi_K(u) = \sup_{v \in K^\circ} \langle u, v \rangle,$$

where $K^\circ = \{u \mid \langle u, v \rangle \leq 0, \forall v \in K\}$ is the polar cone of the given cone $K$. Thus, when $\theta(u) = \psi_K(u)$ we simply take $k \equiv 0$ and $Y = K^\circ$. In our example, $K^\circ = \mathbb{R}^m \times \mathbb{R}^p$. To some extent, this is just a formalism that allows us to claim the classical case as a specialization; however when we take the cone $K$ to be more general than the polyhedral cone used above, we can generate conic programs for example.

The second example involves a piecewise linear function $\theta$: Formally, for $u \in \mathbb{R}$,

$$\theta(u) = \begin{cases} \rho u & \text{if } u \geq 0, \\ \sigma u & \text{else}. \end{cases}$$

In this case, simple calculations prove that $\theta$ has the form ($\theta$ conj) for the choices

$$k \equiv 0 \text{ and } Y = [\sigma, \rho].$$

The special case where $\sigma = -\rho$ results in

$$\theta(u) = \rho |u|.$$  

This allows us to model nonsmooth $L_1$ approximation problems. Another special case results from the choice of $\sigma = -\gamma$, $\rho = 0$, whereby

$$\theta(u) = \gamma \max\{-u, 0\}.$$  

This formulation corresponds to a soft penalization on an inequality constraint, namely if $\theta(-g_1(x))$ is used then nothing is added to the objective function if $g_1(x) \leq 0$, but $\gamma g_1(x)$ is added if the constraint $g_1(x) \leq 0$ is violated. Contrast this to the classical setting above, where $\infty$ is added to the objective if the inequality constraint is violated. It is interesting to see that truncating the set $Y$, which amounts to bounding the multipliers, results in replacing the classical constraint by a linearized penalty.

The third example involves a more interesting choice of $k$. If we wish to replace the "absolute value" penalization given above by a quadratic penalization (as in classical least squares analysis), that is

$$\theta(u) = \gamma u^2$$

then a simple calculation shows that we should take

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = \mathbb{R}.$$

By simply specifying this different choice of $k$ and $Y$ we can generate such models easily and quickly within the modeling system. Note, however, that the reformulation we would use in $\theta(u) = \rho |u|$ and $\theta(u) = \gamma u^2$ are very different as we shall explain in the simple example below. Furthermore, in many applications it has become popular to penalize violations using a
quadratic penalty only within a certain interval, afterwards switching to a linear penalty (chosen to make the penalty function \( \theta \) continuously differentiable - see [7] ). That is:

\[
\theta(u) = \begin{cases} 
\gamma u - \frac{1}{2} \gamma^2 & \text{if } u \geq \gamma \\
\frac{1}{2} u^2 & \text{if } u \in [-\gamma, \gamma] \\
-\gamma u - \frac{1}{2} \gamma^2 & \text{else.}
\end{cases}
\]

Such functions arise from quadratic \( k \) and simple bound sets \( Y \). In particular, the somewhat more general function

\[
\theta(u) = \begin{cases} 
\gamma \beta^2 + \rho (u - \beta) & \text{if } u \geq \beta \\
\gamma u^2 & \text{if } u \in [\alpha, \beta] \\
\gamma \alpha^2 + \sigma (u - \alpha) & \text{else}
\end{cases}
\]

arises from the choice of

\[
k(y) = \frac{1}{4 \gamma} y^2 \quad \text{and} \quad Y = [\sigma, \rho],
\]

with \( \alpha = \frac{\sigma}{2 \gamma} \) and \( \beta = \frac{\rho}{2 \gamma} \).

The final example that we give is that of \( L_\infty \) penalization. This example is different to the examples given above in that \( \theta \) is not separable. However, straightforward calculation can be used to show

\[
\theta(u) = \max_{i=1,\ldots,m} u_i
\]

results from the choice of

\[
k \equiv 0 \quad \text{and} \quad Y = \left\{ y \in \mathbb{R}^m \mid y \geq 0, \sum_{i=1}^m y_i = 1 \right\},
\]

that is, \( Y \) is the unit simplex.

### 9.2 Underlying theory

The underlying structure of \( \theta \) leads to a set of extended optimality conditions and an elegant duality theory. This is based on an extended form of the Lagrangian:

\[
\mathcal{L}(x,y) = f(x) + \sum_{i=1}^m y_i g_i(x) - k(y)
\]

\[x \in X, y \in Y\]

Note that the Lagrangian \( \mathcal{L} \) is smooth - all the nonsmoothness is captured in the \( \theta \) function. The theory is an elegant combination of calculus arguments related to \( g_i \) and its derivatives, and variational analysis for features related to \( \theta \).

It is shown in [9] that under a standard constraint qualification, the first-order conditions of (ENLP) are precisely in the form of the following variational inequality:

\[
\text{VI} \left( \left[ \begin{array}{c} \nabla_x \mathcal{L}(x,y) \\ -\nabla_y \mathcal{L}(x,y) \end{array} \right], X \times Y \right) .
\]

(ENLP VI)

When \( X \) and \( Y \) are simple bound sets, this is simply a complementarity problem.

Note that EMP exploits this result. In particular, if an extended nonlinear program of the form (ENLP) is given to EMP, then the optimality conditions (ENLP VI) are formed as a variational inequality problem and can be processed as outlined above. For a specific example, we cite the fact that if we use the (classical) choice \( k \equiv 0 \) and \( Y = K^c \), then the optimality conditions of (ENLP) are precisely the standard complementarity problem given as (ENLP VI). While this is of interest, we believe that other choices of \( k \) and \( Y \) may be more useful and lead to models that have more practical significance.

Under appropriate convexity assumptions on this Lagrangian, it can be shown that a solution of the VI (ENLP VI) is a saddle point for the Lagrangian on \( X \times Y \). Furthermore, in this setting, the saddle point generates solutions to the primal problem (ENLP) and its dual problem:

\[
\max_{y \in Y} d(y), \quad \text{where} \quad d(y) = \inf_{x \in X} \mathcal{L}(x,y),
\]

with no duality gap.

Unfortunately, the perturbations \( y \) in Rockafellar’s theory are precisely the negative of those used throughout the GAMS system. Thus, we need to replace \( y \) by \(-y\) throughout in the above to recover the same multipliers as those GAMS uses.
9.3 A simple example

As an example, consider the problem

\[
\begin{align*}
\min_{x_1, x_2, x_3} & \quad \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2\max(x_2^2 - 2, 0) \\
\text{s.t.} & \quad x_1/x_2 = \log(x_3), \\
& \quad 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0.
\end{align*}
\]

In this problem, we would take

\[X = \{x \in \mathbb{R}^3 \mid 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0\}.\]

The function \(\theta\) essentially treats 3 separable pieces:

\[
\begin{align*}
g_1(x) &= \log(x_1) - 1, \\
g_2(x) &= x_2^2 - 2, \\
g_3(x) &= x_1/x_2 - \log(x_3).
\end{align*}
\]

A classical problem would force \(g_1(x) = 0, g_2(x) \leq 0\), and \(g_3(x) = 0\), while minimizing \(f(x) = \exp(x_1)\). In our problem, we still force \(g_3(x) = 0\), but apply a (soft) least squares penalty on \(g_1(x)\) and a smaller one-sided penalization on \(g_2(x)\). The above formulation is nonsmooth due to the “max” term in the objective function; in practice we could replace this by:

\[
\begin{align*}
\min_{x_1, x_2, x_3, w} & \quad \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2w \\
\text{s.t.} & \quad x_1/x_2 = \log(x_3), \\
& \quad 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \\
& \quad w \geq x_2^2 - 2, w \geq 0
\end{align*}
\]

and recover a standard form NLP. If the penalty on \(g_1(x)\) would be replaced by a one-norm penalization (instead of least squares), we would have to play a similar game, moving the function \(g_1(x)\) into the constraints and adding additional variable(s). To some extent, this seems unnatural - a modeler should be able to interchange the penalization without having to reformulate the problem from scratch. The proposed extended NLP would not be reformulated at all by the modeler, but allows all these “generalized constraints” to be treated in a similar manner within the modeling system. The actual formulation would take:

\[
\theta(u) = \theta_1(u_1) + \theta_2(u_2) + \theta_3(u_3)
\]

where

\[
\begin{align*}
\theta_1(u_1) &= 5u_1^2, \\
\theta_2(u_2) &= 2\max(u_2, 0), \\
\theta_3(u_3) &= \psi_{[0]}(u_3).
\end{align*}
\]

The discussion above allows us to see that

\[
\begin{align*}
Y &= -(\mathbb{R} \times [0, 2] \times \mathbb{R}), \\
k(y) &= \frac{1}{20}y_1^2 + 0 + 0.
\end{align*}
\]

The corresponding Lagrangian is the smooth function:

\[
\mathcal{L}(x, y) = f(x) - \sum_{i=1}^3 \gamma_i g_i(x) - k(y).
\]

The corresponding VI (ENLP VI) can almost be formulated in GAMS (except that the linear constraint in \(X\) cannot be handled currently except by introducing a \(\theta_4(x)\)). Thus

\[
g_4(x) = 3x_1 + x_2 - 5, \quad \theta_4(u) = \psi_{\mathbb{R}}
\]
resulting in the following choices for \( Y \) and \( k \):

\[
Y = -\left( \mathbb{R} \times [0,2] \times \mathbb{R} \times \mathbb{R}_+ \right),
\]

\[
k(y) = \frac{1}{20} y_1^2 + 0 + 0 + 0.
\]

Since \( X \) and \( Y \) are now simple bound sets, (ENLP VI) is now a complementarity problem and can be solved for example using PATH. A simple "empinfo" file details the choices of \( Y \) and \( k \) from the implemented library:

```
Adjustequ
e1 sqr 5
e2 MaxZ 2
```

### 9.4 Reformulation as a classical NLP

Suppose

\[
\theta(u) = \sup_{y \in Y} \{ \langle u, y \rangle - \frac{1}{2} \langle y, Qy \rangle \}
\]

for a polyhedral set \( Y \in \mathbb{R}^m \) and a symmetric positive semidefinite \( Q \in \mathbb{R}^{m \times m} \) (possibly \( Q = 0 \)).

Suppose further that

\[
X = \{ x \mid Rx \leq r \} ,
Y = \{ y \mid S^Ty \leq s \},
Q = DJ^{-1}D^T ,
F(x) = (g_1(x), \ldots, g_m(x)),
\]

where \( J \) is symmetric and positive definite (for instance \( J = I \)). Then, as outlined by [10], the optimal solutions \( \bar{x} \) of (ENLP) are the \( \bar{x} \) components of the optimal solutions \( (\bar{x}, \bar{z}, \bar{w}) \) to

\[
\min \quad f(x) + \langle s, z \rangle + \frac{1}{2} \langle w, Jw \rangle
\]

s.t. \( Rx \leq r, z \geq 0, F(x) - Sz - Dw = 0. \)

The multiplier on the equality constraint in the usual sense is the multiplier associated with \( \bar{x} \) in the extended Lagrangian for (ENLP). (Note that a Cholesky factorization may be needed to determine \( D \).)

It may be better to solve this reformulated NLP than to solve the VI (ENLP VI). However, it is important that we can convey all types of nonsmooth optimization problems to a solver as smooth optimization problems, and hence it is important to communicate the appropriate structure to the solver interface. We believe that specifying \( Y \) and \( k \) is a theoretically sound way to do this.

### 10 Disjunctive Programs

There are many ways that the EMP model type can be used for further extensions to the modeling capabilities of a given system. In particular, the procedures outlined in [1] for disjunctive programming extensions are also implemented within the EMP model type.

One simple example to highlight this feature is the notion of an ordering of tasks, namely that either job \( i \) completes before job \( j \) starts or the converse, i.e., that the jobs cannot overlap. Such a disjunction can be specified using an empinfo file containing lines:

```
disjunction * seq(i,j) else seq(j,i)
```

In such an example, one can implement a Big-M method, employ indicator constraints, or utilize a convex hull reformulation. The convex hull reformulation is the default strategy; to utilize the Big-M formulation, the additional option

```
default bigm 1000
```
JAMS - an EMP solver

would add binary variables and constraints to impose the disjunction using a Big-M value of 1000. Alternatively, the option setting

```
default indic
```

writes out a model and an option file that implements a reformulation using indicator constraints, which can be handled by the solvers CPLEX, SCIP, and XPRESS. The EMPLIB model `sequence` is sequencing model that implements all of these options.

More complicated (nonlinear) examples make the utility of this approach clearer. The design of a multiproduct batch plan with intermediate storage described in [12] and a synthesis problem involving 8 processes from [11] are also included in the EMP model library. As a final example, the gasoline emission model outlined in [6] is precisely in the form that could exploit the features of EMP related to (nonlinear) disjunctive programming.

11 Empinfo file details

We have explained above many of the standard vectorized ways to write an "empinfo" file. The "empinfo" file has a vectorized format and a more powerful (but more complex) scalar version. We describe some of the features of the scalar version in this section.

An example of the use of the scalar syntax is given below

```plaintext
file info / '%emp.info%' /;
put info / 'equilibrium';
loop(h,
   put / 'min' obj(h) /;
   loop(j, put x(h,j)); put /;
   loop(k, put z(h,k));
   put / costdef(h) /;
   loop(i, put nodebal(h,i)); put /;
);
loop(a, put 'vi', flowdef(a), f(a) /);
```

This is an example of a MOPEC in which the optimization agents are indexed by h and each of these agents control variables x_{ij} and z_{ik}. The objective function for each h is defined in `costdef`, the constraints of each minimization problem are defined in `nodebal`, and the equilibrium constraints that link all the optimization problems are a VI defined by variables f_a and functions within `flowdef`. Sometimes it is necessary to use syntax that generates the "tl" fields of sets, but this seems only necessary when abnormal side conditions are needed.

The format of the empinfo file is given below:

```
Disjunction [chull [big eps] | bigM [big eps threshold] | indic]
   [NOT] var* [NOT] {equ} {ELSEIF [NOT] var* [NOT] {equ}} [ELSE [NOT] {equ}]
Default [chull [big eps] | bigM [big eps threshold] | indic]
ParallelStep1 {equistar}*
AdjustEqu equ abs|sqr|maxz|huber|... {weight {param}}
ModelType MCP|NLP|MIP|...
Bilevel {var} {MAX|MIN obj {var*} {{-eq} equ} {VI {var*} {{-eq} equ var} {{-eq} equ}}
   {DualVar {var [-] equ}}}
Equilibrium {MAX|MIN obj {var*} {{-eq} equ} {VI {var*} {{-eq} equ var} {{-eq} equ}}
   {DualVar {var [-] equ}}}
VI {var*} {{-eq} equ var} {{-eq} equ}
DualEqu {{-eq} equ var}
DualVar {var [-] equ}
```
Bibliography


Least Squares (LS)

Contents

1 Introduction ................................................................. 849
2 Basic Usage ................................................................ 849
3 Options ...................................................................... 850

1 Introduction

There can be difficulties working with linear regression models in GAMS. An explicit minimization problem will be non-linear, as it needs to express a sum of squares. This model may be difficult to solve. Alternatively, it is well known that a linear formulation using the normal equations \((X'X)b=X'y\) will introduce numerical instability.

We have therefore introduced a compact notation where the objective is replaced by a dummy equation: the solver will implicitly understand that we need to minimize the sum of squared residuals. The LS solver will understand this notation and can apply a stable QR decomposition to solve the model quickly and accurately.

2 Basic Usage

A least squares model contains a dummy objective and a set of linear equations:

```gams
sumsq.. sse =n= 0;
fit(i).. data(i,'y') =e= b0 + b1*data(i,'x');
```

```gams
option lp = ls;
model leastsq /fit,sumsq/;
solve leastsq using lp minimizing sse;
```

Here \(sse\) is a free variable that will hold the sum of squared residuals after solving the model. The variables \(b0\) and \(b1\) are the statistical coefficients to be estimated. On return the levels are the estimates and the marginals are the standard errors. The fit equations describe the equation to be fitted.

The constant term or intercept is included in the above example. If you don’t specify it explicitly, and the solver detects the absence of a column of ones in the data matrix \(X\), then a constant term will be added automatically. When you need to do a regression without intercept you will need to use an option `add_constant_term 0`.

It is not needed or beneficial to specify initial values (levels) or an advanced basis (marginals), as they are ignored by the solver.

The estimates are returned as the levels of the variables. The marginals will contain the standard errors. The row levels reported are the residuals errors. In addition a GDX file is written which will contain all regression statistics.

Several complete examples of LS solver usage are available in testlib starting with GAMS Distribution 22.8. For example, model `ls01` takes the data from the Norris dataset found in the NIST collection of statistical reference datasets and reproduces the results and regression statistics found there.
Erwin Kalvelagen is the original author. Further information can be found at Amsterdam Optimization Modeling Group’s web site.

3 Options

The following options are recognized:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxn</td>
<td>Maximum number of cases or observations. This is the number of rows (not counting the dummy objective). When the number of rows is very large, this is probably not a regression problem but a generic LP model. To protect against these cases GAMS does not accept models with an enormous number of rows.</td>
<td>1000</td>
</tr>
<tr>
<td>maxp</td>
<td>Maximum number of coefficients to estimate. This is the number of columns or variables (not counting the dummy objective variable). When the number of variables is very large, this is probably not a regression problem but a generic LP model. To protect against these cases GAMS does not accept models with an enormous number of columns.</td>
<td>25</td>
</tr>
<tr>
<td>add_constant_term</td>
<td>Must be 0, 1, or 2. If the number is zero no constant term or intercept will be added to the problem. If the number is one a constant term will always be added. If the number is two the algorithm will add a constant term only if there is no data column with all ones in the matrix. In this automatic mode, if the user already specified an explicit intercept in the problem, no additional constant term will be added. As the default is two, you will need to add_constant_term 0 in case you want to solve a regression problem without an intercept.</td>
<td>2</td>
</tr>
<tr>
<td>gdx_file_name</td>
<td>Name of the GDX file where results are saved.</td>
<td>ls.gdx</td>
</tr>
</tbody>
</table>
LogMIP

1 Introduction

LogMIP 2.0 is a program for solving linear and nonlinear disjunctive programming problems involving binary variables and disjunction definitions for modeling discrete choices. While the modeling and solution of these disjunctive optimization problems has not yet reached the stage of maturity and reliability as LP, MIP and NLP modeling, these problems have a rich area of applications.

LogMIP 2.0 has been developed by Dr. Aldo Vecchietti from INGAR (Santa Fe, Argentina) and Professor Ignacio E. Grossmann from Carnegie Mellon University (Pittsburgh, USA), and supersedes its previous version, LogMIP 1.0.

- LogMIP 1.0 works with GAMS releases 22.6 (December 2007) to 23.6 (December 2010).
- Starting with GAMS release 23.7, LogMIP 2.0 is an integrated part of a GAMS distribution.
- Changes in version 2.0 are at the level of language, where LogMIP now uses the EMP syntax and modeltype, and at the level of solvers, where LMBigm and LMCHull are combined in the new solver LogMIP.
- Big-M and convex-hull relaxation algorithms for non-linear models are part of LogMIP 2.0.

LogMIP comes free of charge with any licensed GAMS system but needs a subsolver to solve the generated MIP/MINLP models.

For more information see

- Website: http://www.logmip.ceride.gov.ar/
1 Abstract

MILES is a solver for nonlinear complementarity problems and nonlinear systems of equations. This solver can be accessed indirectly through GAMS/MPSGE or GAMS/MCP. This paper documents the solution algorithm, user options, and program output. The purpose of the paper is to provide users of GAMS/MPSGE and GAMS/MCP an overview of how the MCP solver works so that they can use the program effectively.

2 Introduction

MILES is a Fortran program for solving nonlinear complementarity problems and nonlinear systems of equations. The solution procedure is a generalized Newton method with a backtracking line search. This code is based on an algorithm investigated by Mathiesen (1985) who proposed a modeling format and sequential method for solving economic equilibrium models. The method is closely related to algorithms proposed by Robinson (1975), Hogan (1977), Eaves (1978) and Josephy (1979). In this implementation, subproblems are solved as linear complementarity problems (LCPs), using an extension of Lemke’s almost-complementary pivoting scheme in which upper and lower bounds are represented implicitly. The linear solver employs the basis factorization package LUSOL, developed by Gill et al. (1991).

The class of problems for which MILES may be applied are referred to as ”generalized” or ”mixed” complementarity problems, which is defined as follows:
Given: \( F : \mathbb{R}^n \rightarrow \mathbb{R}^n , \ \ell, u \in \mathbb{R}^n \)

Find: \( z, w, v \in \mathbb{R}^n \)

such that \( F(z) = w - v \)

\[ \ell \leq z \leq u, \ \ w \geq 0, \ v \geq 0 \]

\[ w^T (z - \ell) = 0, \quad v^T (u - z) = 0. \]

When \( \ell = -\infty \) and \( u = \infty \) MCP reduces to a nonlinear system of equations. When \( \ell = 0 \) and \( u = +\infty \), the MCP is a nonlinear complementarity problem. Finite dimensional variational inequalities are also MCP. MCP includes inequality-constrained linear, quadratic and nonlinear programs as special cases, although for these problems standard optimization methods may be preferred. MCP models which are not optimization problems encompass a large class of interesting mathematical programs. Specific examples of MCP formulations are not provided here. See Rutherford (1992a) for MCP formulations arising in economics. Other examples are provided by Harker and Pang (1990) and Dirkse (1993).

There are two ways in which a problem may be presented to MILES:

1. MILES may be used to solve computable general equilibrium models generated by MPSGE as a GAMS subsystem. In the MPSGE language, a model-builder specifies classes of nonlinear functions using a specialized tabular input format embedded within a GAMS program. Using benchmark quantities and prices, MPSGE automatically calibrates function coefficients and generates nonlinear equations and Jacobian matrices. Large, complicated systems of nonlinear equations may be implemented and analyzed very easily using this interface to MILES. An introduction to general equilibrium modeling with GAMS/MPSGE is provided by Rutherford (1992a).

2. MILES may be accessed as a GAMS subsystem using variables and equations written in standard GAMS algebra and the syntax for "mixed complementarity problems" (MCP). If more than one MCP solver is available \(^1\), the statement \( \texttt{OPTION MCP} = \texttt{MILES}; \) tells GAMS to use MILES as the MCP solution system. When problems are presented to MILES using the MCP format, the user specifies nonlinear functions using GAMS matrix algebra and the GAMS compiler automatically generates the Jacobian functions. An introduction to the GAMS/MCP modeling format is provided by Rutherford (1992b).

The purpose of this document is to provide users of MILES with an overview of how the solver works so that they can use the program more effectively. Section The Newton Algorithm introduces the Newton algorithm. Section Lemke’s Method with Implicit Bounds describes the implementation of Lemke’s algorithm which is used to solve linear subproblems. Section The Options File defines switches and tolerances which may be specified using the options file. Section Log File Output interprets the run-time log file which is normally directed to the screen. Section Status File Output interprets the status file and the detailed iteration reports which may be generated. Section Termination Messages lists and suggests remedies for abnormal termination conditions.

3 The Newton Algorithm

The iterative procedure applied within MILES to solve nonlinear complementarity problems is closely related to the classical Newton algorithm for nonlinear equations. This first part of this section reviews the classical procedure. A thorough introduction to these ideas is provided by Dennis and Schnabel (1983). For a practical perspective, see Press et al. (1986).

Newton algorithms for nonlinear equations begin with a local (Taylor series) approximation of the system of nonlinear equations. For a point \( \bar{z} \) in the neighborhood of \( \bar{z} \), the system of nonlinear functions is linearized:

\[ LF(z) = F(\bar{z}) + \nabla F(\bar{z})(z - \bar{z}). \]

Solving the linear system \( LF(z) = 0 \) provides the Newton direction from \( \bar{z} \) which given by \( d = -\nabla F^{-1} F(\bar{z}) \).

Newton iteration \( k \) begins at point \( z^k \). First, the linear model formed at \( z^k \) is solved to determine the associated "Newton direction", \( d^k \). Second, a line search in direction \( d^k \) determines the scalar steplength and the subsequent iterate:

\(^1\)There is one other MCP solver available through GAMS: PATH (Ferris and Dirkse,1992), see [PATH vs. MILES](http://www.gams.com/docs/pathvsmiles.htm) for a comparison
An Armijo or "back-tracking" line search initially considers $\lambda = 1$. If $\|F(z^k + \lambda d^k)\| \leq \|F(z^k)\|$, the step size $\lambda$ is adopted, otherwise is multiplied by a positive factor $\alpha$, $0 < \alpha < 1$, and the convergence test is reapplied. This procedure is repeated until either an improvement results or $\lambda < \lambda_0$. When $\lambda = 0$, a positive step is taken provided that $2d^T \alpha \lambda F(z^k + \lambda d^k) < 0$.

Convergence theory for this algorithm is quite well developed. See, for example, Ortega and Rheinbolt (1970) or Dennis and Schnabel (1983). The most attractive aspect of the Newton scheme with the backtracking line search is that in the neighborhood of a well-behaved fixed point, $\lambda = 1$ is the optimal step length and the rate of convergence can be quadratic. If this method finds a solution, it does so very quickly.

The application of Newton methods to nonlinear complementarity problems involves a modification of the search direction. Here, $d$ solves a linear complementarity problem (LCP) rather than a linear system of equations. For iteration $k$, $d$ solves:

$$
F(z^k) + \nabla F(z^k) d - w + v = 0
$$

$$
\ell \leq d + z^k \leq u, \quad w \geq 0, \quad v \geq 0
$$

$$
w^T (d + z^k - \ell) = v^T (u - d - z^k) = 0.
$$

Conceptually, we are solving for $d$, but in practice MILES solves the linear problem in terms of the original variables $z = z^k + d$:

$$
F(z^k) - \nabla F(z^k) z^k + \nabla F(z^k) z = w - v
$$

$$
\ell \leq z \leq u, \quad w \geq 0, \quad v \geq 0
$$

$$
w^T (z - \ell) = 0, \quad v^T (u - z) = 0.
$$

After computing the solution $z$, MILES sets $d^k = z - z^k$.

The linear subproblem incorporates upper and lower bounds on any or all of the variables, assuring that the iterative sequence always remains within the bounds: $\ell \leq z^k \leq u$. This can be helpful when, as is often the case, $F()$ is undefined for some $z \in R^n$.

Convergence of the Newton algorithm applied to MCP hinges on three questions:

1. Does the linearized problem always have a solution?
2. If the linearized problem has a solution, does Lemke’s algorithm find it?
3. Is it possible to show that the computed direction $d^k$ will provide an “improvement” in the solution?

Only for a limited class of functions $F()$ can all three questions be answered in the affirmative. For a much larger class of functions, the algorithm converges in practice but convergence is not “provable”.

The answer to question 3 depends on the choice of a norm by which an improvement is measured. The introduction of bounds and complementarity conditions makes the calculation of an error index more complicated. In MILES, the deviation associated with a candidate solution $z, \varepsilon(z)$, is based on a measure of the extent to which $z, w$ and $v$ violate applicable upper and lower bounds and complementarity conditions.

$^2$ $\alpha$ and $\lambda_0$ correspond to user-specified tolerances ‘DMPFAC’ and ‘MINSTP’, respectively

$^3$ Kaneko (1978) provides some convergence theory for the linearized subproblem
3.1 Evaluating Convergence

Let $\delta^L_i$ and $\delta^U_i$ be indicator variables for whether $z_i$ is off its lower or upper bound. These are defined as:

$$\delta^L_i = \min(1, (z_i - \ell_i)^+) \quad \text{and} \quad \delta^U_i = \min(1, (u_i - z_i)^+) .$$

Given $z$, MILES uses the value of $F(z)$ to implicitly define the slack variables $w$ and $v$:

$$w_i = F_i(z)^+, \quad v_i = \left(-F_i(z)\right)^+ .$$

There are two components to the error term associated with index $i$, one corresponding to $z_i$'s violation of upper and lower bounds:

$$\varepsilon^B_i = (z_i - u_i)^+ + (\ell_i - z_i)^+$$

and another corresponding to violations of complementarity conditions:

$$\varepsilon^C_i = \delta^L_i w_i + \delta^U_i v_i .$$

The error assigned to point $z$ is then taken:

$$\varepsilon(z) = \|\varepsilon^B(z) + \varepsilon^C(z)\|_p$$

for a pre-specified value of $p = 1, 2$ or $+\infty$.

4 Lemke’s Method with Implicit Bounds

A mixed linear complementarity problem has the form:

**Given:** $M \in \mathbb{R}^{n \times n}$, $q, \ell, u \in \mathbb{R}^n$

**Find:** $z, w, v \in \mathbb{R}^n$

such that $Mz + q = w - v,$

$$\ell \leq z \leq u, \quad w \geq 0, \quad v \geq 0, \quad w^T (z - \ell) = 0, \quad v^T (u - z) = 0 .$$

In the Newton subproblem at iteration $k$, the LCP data are given by $q = F(z^k) - \nabla F(z^k) z^k$ and $M = \nabla F(z^k)$.

4.1 The Working Tableau

In MILES, the pivoting scheme for solving the linear problem works with a re-labeled linear system of the form:

$$Bx^B + N \chi^N = q ,$$

where $x^B \in \mathbb{R}^n$, $\chi^N \in \mathbb{R}^{2n}$, and the tableau $[B|N]$ is a conformal "complementary permutation" of $[-M \mid I \mid -I]$. That is, every column $i$ in $B$ must either be the $i$th column of $M, I$ or $-I$, while the corresponding columns $i$ and $i + n$ in $N$ must be the two columns which were not selected for $B$.

4 In the following $x^+ = \max(x, 0)$

5 Parameter $p$ may be selected with input parameter ‘NORM’. The default value for $p$ is $+\infty$. 


To move from the problem defined in terms of \( z, w \) and \( v \) to the problem defined in terms of \( x^B \) and \( x^N \), we assign upper and lower bounds for the \( x^B \) variables as follows:

\[
\begin{align*}
\ell_i, & \quad \text{if } x^B_i = z_i \\
0, & \quad \text{if } x^B_i = w_i \text{ or } v_i,
\end{align*}
\]

\[
\begin{align*}
u_i, & \quad \text{if } x^B_i = z_i \\
\infty, & \quad \text{if } x^B_i = w_i \text{ or } v_i
\end{align*}
\]

The values of the non-basic variables \( x^N_i \) and \( x^N_{i+n} \) are determined by the assignment of \( x^B_i \):

\[
x^B_i = \begin{cases} 
  \ell, & \text{if } \ell_i = 0 \\
  w, & \text{if } w_i = 0 \\
  v, & \text{if } v_i = 0 \\
\end{cases}
\]

In words: if \( z_i \) is basic then both \( w_i \) and \( v_i \) equal zero. If \( z_i \) is non-basic at its lower bound, then \( w_i \) is possibly non-zero and \( v_i \) is non-basic at zero. If \( z_i \) is non-basic at its upper bound, then \( v_i \) is possibly non-zero and \( w_i \) is non-basic at zero.

Conceptually, we could solve the LCP by evaluating \( 3^n \) linear systems of the form:

\[
x^B = B^{-1} \left( q - N x^N \right).
\]

Lemke’s pivoting algorithm provides a procedure for finding a solution by sequentially evaluating some (hopefully small) subsets of these \( 3^n \) alternative linear systems.

### 4.2 Initialization

Let \( B^0 \) denote the initial basis matrix.\(^a\) The initial values for basic variables are then:

\[
\hat{x}^B = (B^0)^{-1} \left( q - N \hat{x}^N \right).
\]

If \( \hat{x}^B \leq \check{x}^B \leq \tilde{x}^B \), then the initial basis is feasible and the complementarity problem is solved.\(^b\) Otherwise, MILES introduces an artificial variable \( z_0 \) and an artificial column \( h \). Basic variables are then expressed as follows:

\[
x^B = \hat{x}^B - \bar{h} z_0,
\]

where \( \bar{h} \) is the “transformed artificial column” (the untransformed column is \( h = B^0 \bar{h} \)). The coefficients of \( \bar{h} \) are selected so that:

1. The values of "feasible" basis variables are unaffected by \( z_0 \): \( \check{x}_i^B \leq \hat{x}_i^B \leq \tilde{x}_i^B \implies \bar{h}_i = 0 \).
2. The “most infeasible” basic variable \( (i = p) \) is driven to its upper or lower bound when \( z_0 = 0 \):

\[
\bar{h}_p = \begin{cases} 
  \check{x}_p^B - \hat{x}_p^B, & \text{if } \check{x}_p^B > \hat{x}_p^B \\
  \hat{x}_p^B - \bar{x}_p^B, & \text{if } \check{x}_p^B < \bar{x}_p^B
\end{cases}
\]

\(^a\)In Miles, \( B^0 \) is chosen using the initially assigned values for \( z \). When \( z_i \leq \ell_i \), then \( x^B_i = w_i \); when \( z_i \geq u_i \), then \( x^B_i = v_i \). otherwise \( x^B_i = z_i \).

\(^b\)The present version of the code simply sets \( B^0 = -I \) and \( x^B = w \) when the user-specified basis is singular. A subsequent version of the code will incorporate the algorithm described by Anstreicher, Lee, and Rutherford [1992] for coping with singularity.
3. All other infeasible basic variables assume values between their upper and lower bounds when $z_0$ increases to 1:

$$x'_i = \begin{cases} 
1 + \frac{x^R_i}{x^L_i} & \text{if } x^L_i > -\infty, \quad x^R_i = +\infty \\
\frac{x^R_i + x^B_i}{2}, & \text{if } x^L_i > -\infty, \quad x^B_i < +\infty \\
x^B_i - 1, & \text{if } x^L_i = -\infty, \quad x^B_i < +\infty.
\end{cases}$$

### 4.3 Pivoting Rules

When $z_0$ enters the basis, it assumes a value of unity, and at this point (baring degeneracy), the subsequent pivot sequence is entirely determined. The entering variable in one iteration is determined by the exiting basic variable in the previous iteration. For example, if $z_i$ were in $B^0$ and introducing $z_0$ caused $z_i$ to move onto its lower bound, then the subsequent iteration introduces $w_i$. Conversely, if $w_i$ were in $B^0$ and $z_0$ caused $w_i$ to fall to zero, the subsequent iteration increases $z_i$ from $\ell_i$. Finally, if $v_i$ were in $B^0$ and $z_0$’s introduction caused $v_i$ to fall to zero, the subsequent iteration decreases $z_i$ from $u_i$.

**Table 1** Pivot Sequence Rules for Lemke’s Algorithm with Implicit Bounds

<table>
<thead>
<tr>
<th>N</th>
<th>Exiting Variable</th>
<th>Entering Variable</th>
<th>Change in Non-basic Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$z_i$ at lower bound</td>
<td>$w_i$ increases from 0</td>
<td>$x^N_i = x^L_i = \ell_i$</td>
</tr>
<tr>
<td>II</td>
<td>$z_i$ at upper bound</td>
<td>$v_i$ increases from 0</td>
<td>$x^N_{i+n} = x^L_i = u_i$</td>
</tr>
<tr>
<td>III</td>
<td>$w_i$ at 0</td>
<td>$z_i$ increases from $\ell_i$</td>
<td>$x^N_i = x^N_{i+n} = 0$</td>
</tr>
<tr>
<td>IV</td>
<td>$v_i$ at 0</td>
<td>$z_i$ decreases from $u_i$</td>
<td>$x^N_i = x^N_{i+n} = 0$</td>
</tr>
</tbody>
</table>

The full set of pivoting rules is displayed in Table 1. One difference between this algorithm and the original Lemke (type III) pivoting scheme (see Lemke (1965), Garcia and Zangwill (1981), or Cottle and Pang (1992)) is that structural variables ($z$’s) may enter or exit the basis at their upper bound values. The algorithm, therefore, must distinguish between pivots in which the entering variable increases from a lower bound from those in which the entering variable decreases from an upper bound.

Another difference with the “usual” Lemke pivot procedure is that an entering structural variable may move from one bound to another. When this occurs, the subsequent pivot introduces the corresponding slack variable. For example, if $\ell_i$ is increased from $\ell_i$ to $u_i$ without driving a basic variable infeasible, then $z_i$ becomes non-basic at $u_i$, and the subsequent pivot introduces $v_i$. This type of pivot may be interpreted as $z_i$ entering and exiting the basis in a single step.

In theory it is convenient to ignore degeneracy, while in practice degeneracy is unavoidable. The present algorithm does not incorporate a lexicographic scheme to avoid cycling, but it does implement a ratio test procedure which assures that when there is more than one candidate, priority is given to the most stable pivot. The admissible set of pivots is determined on both an absolute pivot tolerance ($ZTOLPV$) and a relative pivot tolerance ($ZTOLRP$). No pivot with absolute value smaller than $\min(ZTOLPV, ZTOLRP \|V\|)$ is considered, where $\|V\|$ is the norm of the incoming column.

### 4.4 Termination on a Secondary Ray

Lemke’s algorithm terminates normally when the introduction of a new variable drives $z_0$ to zero. This is the desired outcome, but it does not always happen. The algorithm may be interrupted prematurely when no basic variable “blocks” an incoming variable, a condition known as “termination on a secondary ray”. In anticipation of such outcomes, MILES maintains a record of the value of $z_0$ for successive iterations, and it saves basis information associated with the smallest observed value, $z_0^*$. (In Lemke’s algorithm, the pivot sequence is determined without regard for the effect on $z_0$, and the value of the artificial variable may follow an erratic (non-monotone) path from its initial value of one to the final value of zero.)

---

If all structural variables are subject to finite upper and lower bounds, then no $z_i$ variables may be part of a homogeneous solution adjacent to a secondary ray. This does not imply, however, that secondary rays are impossible when all $z_i$ variables are bounded, as a ray may then be comprised of $w_i$ and $v_i$ variables.
When MILES encounters a secondary ray, a restart procedure is invoked in which the set of basic variables associated with $z_0^*$ are reinstallled. This basis (augmented with one column from the non-basic triplet to replace $z_0$) serves as $B^0$, and the algorithm is restarted. In some cases this procedure permits the pivot sequence to continue smoothly to a solution, while in other cases may only lead to another secondary ray.

## 5 The Options File

MILES accepts the same format options file regardless of how the system is being accessed, through GAMS/MPSGE or GAMS/MCP. The options file is a standard text file which is normally named `MILES.OPT`. The following is a typical options file:

```
BEGIN SPECS
  ITLIMIT = 50
  CONTOL = 1.0E-8
  LUSIZE = 16
END SPECS
```

All entries are of the form "<keyword> = <value>"", where keywords have at most 6 characters. The following are recognized keywords which may appear in the options file, identified by keyword, type and default value, and grouped according to function:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>contol</td>
<td>convergence tolerance Whenever an iterate is encountered for which $\epsilon(z) &lt; \text{CONTOL}$, the algorithm terminates. This corresponds to the GAMS/MINOS parameter &quot;Row tolerance&quot;.</td>
<td>$1e-6$</td>
</tr>
<tr>
<td>dens1</td>
<td>LUSOL: density to start searching maxcol columns The density at which the Markowitz strategy should search maxcol columns and no rows.</td>
<td>$0.3$</td>
</tr>
<tr>
<td>dens2</td>
<td>LUSOL: density to start searching 1 column The density at which the Markowitz strategy should search only 1 column or (preferably) use a dense LU for all the remaining rows and columns.</td>
<td>$0.6$</td>
</tr>
<tr>
<td>dmpfac</td>
<td>damping factor in backtracking linesearch</td>
<td>$0.5$</td>
</tr>
<tr>
<td>elmax1</td>
<td>LUSOL: max multiplier allowed in L during factor</td>
<td>$10$</td>
</tr>
<tr>
<td>elmax2</td>
<td>LUSOL: max multiplier allowed in L during update</td>
<td>$10$</td>
</tr>
<tr>
<td>factim</td>
<td>basis reinversion time Indicates the maximum number of CPU seconds between recalculation of the basis factors.</td>
<td>$0.5$</td>
</tr>
<tr>
<td>invfrq</td>
<td>basis reinversion frequency Determines the maximum number of Lemke iterations between recalculation of the basis factors. This corresponds to the GAMS/MINOS parameter &quot;Factorization frequency&quot;.</td>
<td>$200$</td>
</tr>
<tr>
<td>invlog</td>
<td>toggle Lemke inversion logging A switch which requests LUSOL to generate a report with basis statistics following each refactorization.</td>
<td>$1$</td>
</tr>
<tr>
<td>itch</td>
<td>iteration refinement frequency Indicates the frequency with which the factorization is checked. The number refers to the number of basis replacement operations between refinements. This corresponds to the GAMS/MINOS parameter &quot;Check frequency&quot;.</td>
<td>$25$</td>
</tr>
<tr>
<td>iterlim</td>
<td>minor (Lemke) iterations limit</td>
<td>GAMS IterLim</td>
</tr>
<tr>
<td>itlimt</td>
<td>major (Newton) iterations limit An upper bound on the number of Newton iterations. This corresponds to the GAMS/MINOS parameter &quot;Major iterations&quot;.</td>
<td>$100$</td>
</tr>
</tbody>
</table>

---

When invoking MILES from within GAMS it is possible to use one of several option file names. See the README documentation with GAMS 2.25 for details.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcpdmp</td>
<td>LCP dump (post-scaling) A switch to generate a printout of the LCP data after scaling.</td>
<td>0</td>
</tr>
<tr>
<td>lcpech</td>
<td>LCP echo print (pre-scaling) A switch to generate a printout of the LCP data before scaling, as evaluated.</td>
<td>0</td>
</tr>
<tr>
<td>levout</td>
<td>set output level Sets the level of debug output written to the log and status files. The lowest meaningful value is -1 and the highest is 3. This corresponds, roughly, to the GAMS/MINOS parameter &quot;Print level&quot;.</td>
<td>1</td>
</tr>
<tr>
<td>lprint</td>
<td>LUSOL: print level The print level, &lt; 0 suppresses output. &lt;0 suppresses output 0 gives error messages 1 gives debug output from some of the routines in LUSOL &gt;=2 gives the pivot row and column and the no. of rows and columns involved at each elimination step in lu1fac</td>
<td>0</td>
</tr>
<tr>
<td>lusize</td>
<td>LUSOL: multiplier for estimating memory requirements Used to estimate the number of LU nonzeros which will be stored, as a multiple of the number of nonzeros in the Jacobian matrix.</td>
<td>5</td>
</tr>
<tr>
<td>maxcol</td>
<td>LUSOL: max cols to search for pivot element In lu1fac is the maximum number of columns searched allowed in a Markowitz type search for the next pivot element. For some of the factorization, the number of rows searched is maxrow = maxcol - 1.</td>
<td>5</td>
</tr>
<tr>
<td>maxeul</td>
<td>Euler iteration limit</td>
<td>0</td>
</tr>
<tr>
<td>mineul</td>
<td>Euler iteration limit</td>
<td>0</td>
</tr>
<tr>
<td>minstp</td>
<td>minimum step length</td>
<td>0.01</td>
</tr>
<tr>
<td>norm</td>
<td>norm to use in measuring deviation Defines the vector norm to be used for evaluating epsilon(z). Acceptable values are 1, 2 or 3 which correspond to p = 1, 2 and +INF.</td>
<td>1</td>
</tr>
<tr>
<td>nrsmax</td>
<td>restart limit Sets an upper bound on the number of restarts which the linear subproblem solver will undertake before giving up.</td>
<td>1</td>
</tr>
<tr>
<td>pivlog</td>
<td>toggle Lemke pivot logging A switch to generate a status file listing of the Lemke pivot sequence.</td>
<td>0</td>
</tr>
<tr>
<td>plinfy</td>
<td>infinity used by the solver The value assigned for &quot;plus infinity&quot; (&quot;+INF&quot; in GAMS notation).</td>
<td>1e20</td>
</tr>
<tr>
<td>scale</td>
<td>turn on scaling at every iteration Invokes row and column scaling of the LCP tableau in every iteration. This corresponds, roughly, to the GAMS/MINOS switch &quot;scale all variables&quot;.</td>
<td>1</td>
</tr>
<tr>
<td>small</td>
<td>LUSOL: absolute zero tolerance The absolute tolerance for treating reals as zero.</td>
<td>3.0d-13</td>
</tr>
<tr>
<td>spdadj</td>
<td>Euler adjustment factor</td>
<td>0.01</td>
</tr>
<tr>
<td>uspace</td>
<td>LUSOL: factor limiting waste space in U In lu1fac, the row or column lists are compressed if their length exceeds uspace times the length of either file after the last compression.</td>
<td>3</td>
</tr>
<tr>
<td>utol1</td>
<td>LUSOL: absolute tolerance U diagonal The absolute tol for flagging small diagonals of U.</td>
<td>3.7d-11</td>
</tr>
<tr>
<td>utol2</td>
<td>LUSOL: relative tolerance U diagonal The relative tol for flagging small diagonals of U.</td>
<td>3.7d-11</td>
</tr>
<tr>
<td>ztolda</td>
<td>zero tolerance on coefficients</td>
<td></td>
</tr>
</tbody>
</table>
6 Log File Output

The log file is intended for display on the screen in order to permit monitoring progress. Relatively little output is generated. A sample iteration log is displayed in Table 2. This output is from two cases solved in succession. This and subsequent output comes from program TRNSP.FOR which calls the MILES library directly. (When MILES is invoked from within GAMS, at most one case is processed at a time.)

The first line of the log output gives the MILES program date and version information. This information is important for bug reports.

The line beginning "Work space..." reports the amount of memory which has been allocated to solve the model - 10K for this example. Thereafter is reported the initial deviation together with the name of the variable associated with the largest imbalance ($\varepsilon_B + \varepsilon_C$). The next line reports the convergence tolerance.

The lines beginning 0 and 1 are the major iteration reports for those iterations. the number following the iteration number is the current deviation, and the third number is the Armijo step length. The name of the variable complementary to the equation with the largest associated deviation is reported in parenthesis at the end of the line.

Following the final iteration is a summary of iterations, refactorizations, and final deviation. The final message reports the solution status. In this case, the model has been successfully processed ("Solved.").

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ztolpv</td>
<td>Absolute zero tolerance on pivots</td>
<td>The absolute pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter &quot;Pivot tolerance&quot; as it applies for nonlinear problems.</td>
</tr>
<tr>
<td>ztolrp</td>
<td>Relative zero tolerance on pivots</td>
<td>The relative pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter &quot;Pivot tolerance&quot; as it applies for nonlinear problems.</td>
</tr>
<tr>
<td>ztolz0</td>
<td>Absolute tolerance for installing cover elements</td>
<td>$1e^{-6}$</td>
</tr>
<tr>
<td>ztolze</td>
<td>Feasibility tolerance</td>
<td>Used in the subproblem solution to determine when any variable has exceeded an upper or lower bound. This corresponds to GAMS/MINOS parameter &quot;Feasibility tolerance&quot;.</td>
</tr>
</tbody>
</table>

Table 2 Sample Iteration Log

MILES (July 1993) Ver:225-386-02

Thomas F. Rutherford
Department of Economics
University of Colorado

Technical support available only by Email: TOM@GAMS.COM

Work space allocated -- 0.01 Mb

Initial deviation ......... 3.250E+02 P_01
Convergence tolerance .... 1.000E-06

0 3.25E+02 1.00E+00 (P_01 )
1 1.14E-13 1.00E+00 (W_02 )

Major iterations ......... 1
Lemke pivots .............. 10
Refactorizations ........... 2
Deviation ................. 1.137E-13
**862 MILES**

Solved.

Work space allocated -- 0.01 Mb

Initial deviation ........ 5.750E+02 W_02
Convergence tolerance .... 1.000E-06

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.75E+02</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>1</td>
<td>2.51E+01</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>2</td>
<td>4.53E+00</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>3</td>
<td>1.16E+00</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>4</td>
<td>3.05E-01</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>5</td>
<td>8.08E-02</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>6</td>
<td>2.14E-02</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>7</td>
<td>5.68E-03</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>8</td>
<td>1.51E-03</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>9</td>
<td>4.00E-04</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>10</td>
<td>1.06E-04</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>11</td>
<td>2.82E-05</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>12</td>
<td>7.47E-06</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>13</td>
<td>1.98E-06</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>14</td>
<td>5.26E-07</td>
<td>1.00E+00</td>
</tr>
</tbody>
</table>

Major iterations ........ 14
Lemke pivots ............ 23
Refactorizations ........ 15
Deviation ............... 5.262E-07

Solved.

**7 Status File Output**

The status file reports more details regarding the solution process than are provided in the log file. Typically, this file is written to disk and examined only if a problem arises. Within GAMS, the status file appears in the listing only following the GAMS statement `OPTION SYSOUT=ON;`.

The level of output to the status file is determined by the options passed to the solver. In the default configuration, the status file receives all information written to the log file together a detailed listing of all switches and tolerances and a report of basis factorization statistics.

When output levels are increased from their default values using the options file, the status file can receive considerably more output to assist in debugging. `Table 3 - Table 6` present a status file generated with `LEVOUT=3` (maximum), `PIVLOG=T`, and `LCPECH=T`.

The status file begins with the same header as the log file. Thereafter is a complete echo-print of the user-supplied option file when one is provided. Following the core allocation report is a full echo-print of control parameters, switches and tolerance as specified for the current run.

`Table 4` continues the status file. The iteration-by-iteration report of variable and function values is produced whenever `LEVOUT >= 2`. `Table 4` also contains an LCP echo-print. This report has two sections: `$ROWS` and `$COLUMNS`. The four columns of numbers in the `$ROWS` section are the constant vector (q), the current estimate of level values for the associated variables (z), and the lower and upper bounds vectors (ℓ and u). The letters L and U which appear between the ROW and Z columns are used to identify variables which are non-basic at their lower and upper bounds, respectively. In this example, all upper bounds equal $+\infty$, so no variables are non-basic at their upper bound.

By convention, only variable (and not equation names) appear in the status file. An equation is identified by the corresponding variable. We therefore see in the `$COLUMNS`: section of the matrix echo-print, the row names correspond to the names of z.
variables. The names assigned to variables $z_i$, $w_i$ and $v_i$ are $z_{\text{name } i}$, $w_{\text{name } i}$, and $v_{\text{name } i}$, as shown in the $\text{COLUMNS}$ section. The nonzeros for $w_{\text{<>}}$ and $v_{\text{<>}}$ variables are not shown because they are assumed to be $-\text{/}+I$.

The status file output continues on Table 5 where the first half of the table reports output from the matrix scaling procedure, and the second half reports the messages associated with initiation of Lemke’s procedure.

The "lu6chk warning” is a LUSOL report. Thereafter are two factorization reports. Two factorizations are undertaken here because the first basis was singular, so the program install all the lower bound slacks in place of the matrix defined by the initial values, $z$.

Following the second factorization report, at the bottom of Table 5 is a summary of initial pivot. "Infeasible in 3 rows.” indicates that $\bar{h}$ contains 3 nonzero elements. "Maximum infeasibility” reports the largest amount by which a structural variable violates an upper or lower bound. ”Artificial column with 3 elements.” indicates that the vector $h = B^0\bar{h}$ contains 3 elements (note that in this case $B^0 = -I$ because the initial basis was singular, hence the equivalence between the number of nonzeros in $\bar{h}$ and $h$).

Table 6 displays the final section of the status file. At the top of the table is the Lemke iteration log. The columns are interpreted as follows:

- **ITER** is the iteration index beginning with 0.
- **STATUS** is a statistic representing the efficiency of the Lemke path. Formally, status is the ratio of the minimum number of pivots from $B_0$ to the current basis divided by the actual number of pivots. When the status is 1, Lemke’s algorithm is performing virtually as efficiently as a direct factorization (apart from the overhead of basis factor updates.)
- **Z\%** indicates the percentage of columns in the basis are "structural" ( $z$’s).
- **Z0** indicates the value of the artificial variable. Notice that in this example, the artificial variable declines monotonically from its initial value of unity.
- **ERROR** is a column in which the factorization error is reported, when it is computed. For this run, ITCH=30 and hence no factorization errors are computed.
- **INFEAS** is a column in which the magnitude of the infeasibility introduced by the artificial column (defined using the box-norm) is reported. (In MILES the cover vector $h$ contains many different nonzero values, not just 1’s; so there may be a large difference between the magnitude of the artificial variable and the magnitude of the induced infeasibility.
- **PIVOTS** reports the pivot magnitude in both absolute terms (the first number) and relative terms (the second number). The relative pivot size is the ratio of the pivot element to the norm of the incoming column.
- **IN/OUT** report the indices (not names) of the incoming and outgoing columns for every iteration. Notice that Lemke’s iteration log concludes with variable $z_0$ exiting the basis.

The convergence report for iteration 1 is no different from the report written to the log file, and following this is a second report of variable and function values. We see here that a solution has been obtained following a single subproblem. This is because the underlying problem is, in fact, linear.

The status file (for this case) concludes with an iteration summary identical to the log file report and a summary of how much CPU time was employed overall and within various subtasks. (Don’t be alarmed if the sum of the last five numbers does not add up to the first number - some cycles are not monitored precisely.)

Table 3 Status File with Debugging Output (page 1 of 4)
User supplied option file:

>BEGIN
> PIVLOG = .TRUE.
> LCPECH = .TRUE.
> LEVOUT = 3
>END

Work space allocated -- 0.01 Mb

NEWTON algorithm control parameters:
- Major iteration limit .. (ITLIMT) : 25
- Damping factor .. (DMPFAC) : 5.00E-01
- Minimum step length .. (MINSTP) : 1.00E-02
- Norm for deviation .. (NORM) : 3
- Convergence tolerance .. (CONTOL) : 1.000E-06

LEMKE algorithm control parameters:
- Iteration limit .. (ITERLIM) : 1000
- Factorization frequency .. (INVFRQ) : 200
- Feasibility tolerance .. (ZTOLZE) : 1.000E-06
- Coefficient tolerance .. (ZTOLDA) : 1.483E-08
- Abs. pivot tolerance .. (ZTOLPV) : 3.644E-11
- Rel. pivot tolerance .. (ZTOLRP) : 3.644E-11
- Cover vector tolerance .. (ZTOLZ0) : 1.000E-06
- Scale every iteration .. (SCALE) : T
- Restart limit .. (NRSMAX) : 1

Output control switches:
- LCP echo print .. (LCPECH) : F
- LCP dump .. (LCPDMP) : T
- Lemke inversion log .. (INVLOG) : T
- Lemke pivot log .. (PIVLOG) : T

Initial deviation : 3.250E+02  P_01
Convergence tolerance : 1.000E-06

Convergence Report, Iteration 0

<table>
<thead>
<tr>
<th>ITER</th>
<th>DEVIATION</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.25E+02</td>
<td>1.00E+00 (P_01)</td>
</tr>
</tbody>
</table>

======================================================================
Convergence Report, Iteration 0
Table 4 Status File with Debugging Output (page 2 of 4)

<table>
<thead>
<tr>
<th>Iteration 0 values.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ROW</strong></td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>$X_{01}$</td>
</tr>
<tr>
<td>$X_{02}$</td>
</tr>
<tr>
<td>$X_{03}$</td>
</tr>
<tr>
<td>$X_{02}$</td>
</tr>
<tr>
<td>$X_{01}$</td>
</tr>
<tr>
<td>$X_{02}$</td>
</tr>
<tr>
<td>$W_01$</td>
</tr>
<tr>
<td>$W_02$</td>
</tr>
<tr>
<td>$P_01$</td>
</tr>
<tr>
<td>$P_02$</td>
</tr>
<tr>
<td>$P_03$</td>
</tr>
</tbody>
</table>

Function Evaluation, Iteration: 0

$\text{RGWS:}$

| $X_{01}$ | $-2.250000E-01$ | $0.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |
| $X_{02}$ | $-1.530000E-01$ | $0.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |
| $X_{03}$ | $-1.619999E-01$ | $0.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |
| $X_{02}$ | $-2.250000E-01$ | $0.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |
| $X_{01}$ | $-1.619999E-01$ | $0.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |
| $W_01$ | $-3.250000E+02$ | $0.000000E+00$ | $0.000000E+00$ | $1.000000E+00$ |
| $W_02$ | $-5.750000E+02$ | $0.000000E+00$ | $0.000000E+00$ | $1.000000E+00$ |
| $P_01$ | $3.250000E+02$ | $1.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |
| $P_02$ | $3.000000E+02$ | $1.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |
| $P_03$ | $2.750000E+02$ | $1.000000E+00$ | $0.000000E+00$ | $1.000000E+20$ |

$\text{COLUMNS:}$

| $Z-X_{01}$ | $W_01$ | $-1.000000E+00$ |
| $Z-X_{02}$ | $P_01$ | $1.000000E+00$ |
| $Z-X_{03}$ | $P_02$ | $1.000000E+00$ |
| $Z-X_{01}$ | $W_02$ | $-1.000000E+00$ |
| $Z-X_{02}$ | $P_03$ | $1.000000E+00$ |
| $Z-X_{02}$ | $W_02$ | $-1.000000E+00$ |
| $Z-X_{03}$ | $P_01$ | $1.000000E+00$ |
| $Z-X_{02}$ | $W_02$ | $-1.000000E+00$ |
| $Z-X_{01}$ | $P_03$ | $1.000000E+00$ |
| $Z-X_{02}$ | $W_02$ | $-1.000000E+00$ |
| $Z-P_01$ | $X_{01}$ | $1.000000E+00$ |
| $Z-P_02$ | $X_{02}$ | $1.000000E+00$ |
| $Z-P_03$ | $X_{01}$ | $-1.000000E+00$ |
Table 5 Status File with Debugging Output (page 3 of 4)

SCALING LCP DATA

--------

<table>
<thead>
<tr>
<th>MIN ELEM</th>
<th>MAX ELEM</th>
<th>MAX COL RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFTER 0</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>AFTER 1</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>AFTER 2</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
</tr>
<tr>
<td>AFTER 3</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
</tr>
</tbody>
</table>

SCALING RESULTS:

\[ A(I,J) \leq A(I,J) \times R(I) / C(J) \]

ROW | ROW | Z COLUMN | W COLUMN | V COLUMN
---|-----|----------|----------|----------
1   | 1.0000 | 1.0000  | 1.0000  | 1.0000
2   | 1.0000 | 1.0000  | 1.0000  | 1.0000
3   | 1.0000 | 1.0000  | 1.0000  | 1.0000
4   | 1.0000 | 1.0000  | 1.0000  | 1.0000
5   | 1.0000 | 1.0000  | 1.0000  | 1.0000
6   | 1.0000 | 1.0000  | 1.0000  | 1.0000
7   | 1.0000 | 1.0000  | 1.0000  | 1.0000
8   | 1.0000 | 1.0000  | 1.0000  | 1.0000
9   | 1.0000 | 1.0000  | 1.0000  | 1.0000
10  | 1.0000 | 1.0000  | 1.0000  | 1.0000
11  | 1.0000 | 1.0000  | 1.0000  | 1.0000

lu6chk warning. The matrix appears to be singular.

nrank = 8  rank of U
n - nrank = 3  rank deficiency
nsing = 3  singularities
jsing = 10  last singular column
dumax = 1.00E+00  largest triangular diag
dumin = 1.00E+00  smallest triangular diag

LUSOL 5.4 FACTORIZATION STATISTICS

Compressns 0  Merit 0.00  LenL 0  LenU 14
Increase 0.00  M 11  UT 11  D1 0
Lmax 0.0E+00  Bmax 1.0E+00  Umax 1.0E+00  Umin 1.0E+00
Growth 1.0E+00  LT 0  BP 0  D2 0

LUSOL 5.4 FACTORIZATION STATISTICS

Compressns 0  Merit 0.00  LenL 0  LenU 11
Increase 0.00  M 11  UT 11  D1 0
Lmax 0.0E+00  Bmax 1.0E+00  Umax 1.0E+00  Umin 1.0E+00
Growth 1.0E+00  LT 0  BP 0  D2 0

CONSTRUCTING ARTIFICIAL COLUMN

--- Infeasible in 3 rows.
--- Maximum infeasibility: 3.250E+02
--- Artificial column with 3 elements.
--- Pivoting in row: 9 to replace column 20
--- Pivot element: -3.250E+02
### LEMKE PIVOT STEPS

<table>
<thead>
<tr>
<th>ITER</th>
<th>STATUS</th>
<th>Z%</th>
<th>Z0</th>
<th>ERROR</th>
<th>INFEAS.</th>
<th>PIVOTS</th>
<th>IN</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>0</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1</td>
<td>Z0</td>
<td>W 9</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>9</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>2</td>
<td>Z</td>
<td>9 W 1</td>
</tr>
<tr>
<td>3</td>
<td>1.00</td>
<td>18</td>
<td>0.997</td>
<td>9.0E-01</td>
<td>9.0E-01</td>
<td>1</td>
<td>Z</td>
<td>1 W 10</td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>27</td>
<td>0.997</td>
<td>1.0E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>Z</td>
<td>10 W 2</td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>36</td>
<td>0.996</td>
<td>9.0E-01</td>
<td>4.0E-01</td>
<td>1</td>
<td>Z</td>
<td>2 W 11</td>
</tr>
<tr>
<td>6</td>
<td>1.00</td>
<td>45</td>
<td>0.996</td>
<td>1.0E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>Z</td>
<td>2 W 11</td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>55</td>
<td>0.479</td>
<td>2.0E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>Z</td>
<td>6 W 7</td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>64</td>
<td>0.479</td>
<td>1.0E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>Z</td>
<td>7 W 4</td>
</tr>
<tr>
<td>9</td>
<td>1.00</td>
<td>73</td>
<td>0.000</td>
<td>1.0E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>Z</td>
<td>4 W 8</td>
</tr>
<tr>
<td>10</td>
<td>1.00</td>
<td>73</td>
<td>0.000</td>
<td>1.0E+00</td>
<td>1.0E+00</td>
<td>1</td>
<td>Z</td>
<td>8 Z0</td>
</tr>
</tbody>
</table>

---

**Convergence Report, Iteration 1**

**Iteration 1 values.**

<table>
<thead>
<tr>
<th>ROW</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_01_01</td>
<td>2.50000E+01</td>
<td>-8.32667E-17</td>
</tr>
<tr>
<td>X_01_02</td>
<td>3.00000E+02</td>
<td>-5.55112E-17</td>
</tr>
<tr>
<td>X_01_03</td>
<td>0.00000E+00</td>
<td>3.60000E-02</td>
</tr>
<tr>
<td>X_02_01</td>
<td>3.00000E+02</td>
<td>-8.32667E-17</td>
</tr>
<tr>
<td>X_02_02</td>
<td>0.00000E+00</td>
<td>8.99999E-03</td>
</tr>
<tr>
<td>X_02_03</td>
<td>2.75000E+02</td>
<td>2.77556E-17</td>
</tr>
<tr>
<td>W_01</td>
<td>1.00000E+00</td>
<td>-1.13687E-13</td>
</tr>
<tr>
<td>W_02</td>
<td>1.00000E+00</td>
<td>1.13687E-13</td>
</tr>
<tr>
<td>P_01</td>
<td>1.22500E+00</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>P_02</td>
<td>1.15300E+00</td>
<td>0.00000E+00</td>
</tr>
<tr>
<td>P_03</td>
<td>1.12600E+00</td>
<td>0.00000E+00</td>
</tr>
</tbody>
</table>

---

Major iterations ........ 1  
Lemke pivots ............ 10  
Refactorizations ........ 2  
Deviation ............... 1.137E-13  
Solved.

Total solution time ..: 0.6 sec.  
Function & Jacobian.: 0.2 sec.  
LCP solution .........: 0.2 sec.  
Refactorizations ....: 0.1 sec.  
FTRAN ..............: 0.0 sec.  
Update ............: 0.1 sec.
8 Termination Messages

Basis factorization error in INVERT. An unexpected error code returned by LUSOL. > This should normally not occur. Examine the status file for a message from LUSOL 10.

Failure to converge. Two successive iterates are identical - the Newton search direction is not defined. This should normally not occur.

Inconsistent parameters ZTOLZ0, ZTOLZE. ZTOLZ0 determines the smallest value loaded into the cover vector h, whereas ZTOLZE is the feasibility tolerance employed in the Harris pivot selection procedure. If ZTOLZ0 < ~ZTOLZE, Lemke's algorithm cannot be executed because the initial basis is infeasible.

Insufficient space for linearization. Available memory is inadequate for holding the nonzeros in the Jacobian. More memory needs to be allocated. On a PC, you probably will need to install more physical memory - if there is insufficient space for the Jacobi matrix, there is far too little memory for holding the LU factors of the same matrix.

Insufficient space to invert. More memory needs to be allocated for basis factors. Increase the value of LUSIZE in the options file, or assign a larger value to <model>.workspace if MILES is accessed through GAMS.

Iteration limit exceeded. This can result from either exceeding the major (Newton) or minor (Lemke) iterations limit. When MILES is invoked from GAMS, the Lemke iteration limit can be set with the statement <model>.iterlim = xx;} (the default value is 1000). The Newton iteration limit is 25 by default, and it can be modified only through the ITLIMT option.

Resource interrupt. Elapsed CPU time exceeds options parameter RESLIM. To increase this limit, either add RESLIM = xxx in the options file or (if MILES is invoked from within GAMS), add a GAMS statement <model>.RESLIM = xxx;.

Singular matrix encountered. Lemke’s algorithm has been interrupted due to a singularity arising in the basis factorization, either during a column replacement or during a refactorization. For some reason, a restart is not possible.

Termination on a secondary ray. Lemke’s algorithm terminated on a secondary ray. For some reason, a restart is not possible.

Unknown termination status. The termination status flag has not been set, but the code has interrupted. Look in the status file for a previous message. This termination code should not happen often.

9 References

S. Dirkse "Robust solution of mixed complementarity problems", Computer Sciences Department, University of Wisconsin (1992).

10 Within GAMS, insert the line "OPTION SYSOUT=ON;" prior to the solve statement and resubmit the program in order to pass the MILES solver status file through to the listing.


In this file, Dantzig’s original transportation model is reformulated as a linear complementarity problem. We first solve the model with fixed demand and supply quantities, and then we incorporate price-responsiveness on both sides of the market.

T.Rutherford 3/91 (revised 5/91)

This problem finds a least cost shipping schedule that meets requirements at markets and supplies at factories

References:


SETS
I canning plants / SEATTLE, SAN-DIEGO /
J markets / NEW-YORK, CHICAGO, TOPEKA /;

PARAMETERS
A(I) capacity of plant i in cases (when prices are unity)
/ SEATTLE 325
    SAN-DIEGO 575 /,
B(J) demand at market j in cases (when prices equal unity)
/ NEW-YORK 325
    CHICAGO 300
    TOPEKA 275 /,
ESUB(J) Price elasticity of demand (at prices equal to unity)
/ NEW-YORK 1.5
    CHICAGO 1.2
    TOPEKA 2.0 /

TABLE D(I,J) distance in thousands of miles

<table>
<thead>
<tr>
<th></th>
<th>NEW-YORK</th>
<th>CHICAGO</th>
<th>TOPEKA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEATTLE</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>SAN-DIEGO</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

SCALAR F freight in dollars per case per thousand miles /90/;

PARAMETER C(I,J) transport cost in thousands of dollars per case;

\[ C(I,J) = F \times D(I,J) / 1000; \]
PARAMETER PBAR(J) Reference price at demand node J;

Table 8 Transport Model in GAMS/MCP (page 2 of 2)

POSITIVE VARIABLES
  W(I) shadow price at supply node i,
  P(J) shadow price at demand node j,
  X(I,J) shipment quantities in cases;

EQUATIONS
  SUPPLY(I) supply limit at plant i,
  FXDEMAND(J) fixed demand at market j,
  PRDEMAND(J) price-responsive demand at market j,
  PROFIT(I,J) zero profit conditions;

PROFIT(I,J.. W(I) + C(I,J) =G= P(J);
SUPPLY(I.. A(I) =G= SUM(J, X(I,J));
FXDEMAND(J.. SUM(I, X(I,J)) =G= B(J);
PRDEMAND(J.. SUM(I, X(I,J)) =G= B(J) * (PBAR(J)/P(J))**ESUB(J);

* Declare models including specification of equation-variable association:
MODEL FIXEDQTY / PROFIT.X, SUPPLY.W, FXDEMAND.P/ ;
MODEL EQUILQTY / PROFIT.X, SUPPLY.W, PRDEMAND.P/ ;

* Initial estimate:
  P.L(J) = 1;  W.L(I) = 1;
PARAMETER REPORT(*,*,*) Summary report;
SOLVE FIXEDQTY USING MCP;
REPORT("FIXED",I,J) = X.L(I,J);  REPORT("FIXED","Price",J) = P.L(J);
REPORT("FIXED",I,"Price") = W.L(I);

* Calibrate the demand functions:
  PBAR(J) = P.L(J);
* Replicate the fixed demand equilibrium:
SOLVE EQUILQTY USING MCP;
REPORT("EQUIL",I,J) = X.L(I,J);  REPORT("EQUIL","Price",J) = P.L(J);
REPORT("EQUIL",I,"Price") = W.L(I);
DISPLAY "BENCHMARK CALIBRATION", REPORT;

* Compute a counter-factual equilibrium:
C("SEATTLE","CHICAGO") = 0.5 * C("SEATTLE","CHICAGO");
SOLVE FIXEDQTY USING MCP;
REPORT("FIXED",I,J) = X.L(I,J); REPORT("FIXED","Price",J) = P.L(J);
REPORT("FIXED",I,"Price") = W.L(I);

* Replicate the fixed demand equilibrium:

SOLVE EQUILQTY USING MCP;
REPORT("EQUIL",I,J) = X.L(I,J); REPORT("EQUIL","Price",J) = P.L(J);
REPORT("EQUIL",I,"Price") = W.L(I);

DISPLAY "Reduced Seattle-Chicago transport cost:", REPORT;
1 Introduction

The GAMS/NLPEC solver, developed jointly by Michael Ferris of UW-Madison and GAMS Development, solves MPEC and MCP models via reformulation of the complementarity constraints. The resulting sequence of NLP models are parameterized by a scalar $\mu$ and solved by existing NLP solvers. The resulting solutions used to recover an MPEC or MCP solution.

GAMS/NLPEC serves a number of purposes. In many cases, it is an effective tool for solving MPEC models, the only such tool available within GAMS. It also serves as a way to experiment with the many reformulation strategies proposed for solving MPEC and MCP models. Without something like NLPEC (and a library of models to test with) a comprehensive and thorough test and comparison of the various reformulation strategies would not be possible. To better serve these purposes, NLPEC has an open architecture. The model reformulations are written out as GAMS source for solution via an NLP solver, so it is possible to view this source and modify it if desired.

A brief note about notation is in order. The GAMS keyword positive is used to indicate nonnegative variables. The same holds for nonpositive variables and the GAMS keyword negative.

2 Usage

GAMS/NLPEC can solve models of two types: MPEC and MCP. If you did not specify NLPEC as the default MPEC or MCP solver, use the following statement in your GAMS model before the solve statement:

```
option MPEC=nlpec; { or MCP }
```

You can also make NLPEC the default solver via the command line:
You can use NLPEC with its default strategy and formulation, but most users will want to use an options file (Section Options) after reading about the different types of reformulations possible (Section Reformulation). In addition, an understanding of the architecture of NLPEC (Section Open Architecture) will be helpful in understanding how GAMS options are treated. Although NLPEC doesn’t use the GAMS options `workspace`, `workfactor`, `optcr`, `optca`, `reslim`, `iterlim`, and `domlim` directly, it passes these options on in the reformulated model so they are available to the NLP subsolver.

3 Reformulation

In this section we describe the different ways that the NLPEC solver can reformulate an MPEC as an NLP. The description also applies to MCP models - just consider MCP to be an MPEC with a constant objective. The choice of reformulation, and the subsidiary choices each reformulation entails, are controlled by the options (see Section Setting the Reformulation Options) mentioned throughout this section.

The original MPEC model is given as:

\[
\min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} f(x, y)
\]

subject to the constraints

\[
g(x, y) \leq 0
\]

and

\[
y \text{ solves } \text{MCP}(h(x, \cdot), \mathbf{B}).
\]

In most of the reformulations, the objective function (1) is included in the reformulated model without change. In some cases, it may be augmented with a penalty function. The variables $x$ are typically called upper level variables (because they are associated with the upper level optimization problem) whereas the variables $y$ are sometimes termed lower level variables.

The constraints (2) are standard nonlinear programming constraints specified in GAMS in the standard fashion. In particular, these constraints may be less than inequalities as shown above, or equalities or greater than inequalities. The constraints will be unaltered by all our reformulations. These constraints may involve both $x$ and $y$, or just $x$ or just $y$, or may not be present at all in the problem.

The constraints of interest are the equilibrium constraints (3), where (3) signifies that $y \in \mathbb{R}^m$ is a solution to the mixed complementarity problem (MCP) defined by the function $h(x, \cdot)$ and the box $\mathbf{B}$ containing (possibly infinite) simple bounds on the variables $y$. A point $y$ with $a_i \leq y_i \leq b_i$ solves (3) if for each $i$ at least one of the following holds:

\[
\begin{align*}
h_i(x, y) &= 0 \\
h_i(x, y) &> 0, y_i = a_i; \\
h_i(x, y) &< 0, y_i = b_i.
\end{align*}
\]

As a special case of (4), consider the case where $a = 0$ and $b = +\infty$. Since $y_i$ can never be $+\infty$ at a solution, (4) simplifies to the nonlinear complementarity problem (NCP):

\[
0 \leq h_i(x, y), 0 \leq y_i \text{ and } y_i h_i(x, y) = 0, i = 1, \ldots, m
\]

namely that $h$ and $y$ are nonnegative vectors with $h$ perpendicular to $y$. This motivates our shorthand for (4), the ”perp to” symbol $\perp$:

\[
h_i(x, y) \perp y_i \in [a_i, b_i]
\]

The different ways to force (6) to hold using (smooth) NLP constraints are the basis of the NLPEC solver.

We introduce a simple example now that we will use throughout this document for expositional purposes:

\[
\begin{align*}
\min_{x_1, x_2, y_1, y_2} &\quad x_1 + x_2 \\
\text{subject to} &\quad x_1^2 + x_2^2 \leq 1 \\
&\quad y_1 - y_2 + 1 \leq x_1 \perp y_1 \geq 0 \\
&\quad x_2 + y_2 \perp y_2 \in [-1, 1]
\end{align*}
\]
This problem has the unique solution \( x_1 = 0, x_2 = -1, y_1 = 0, y_2 = 1 \). Note that \( f(x,y) = x_1 + x_2 \) and \( g(x,y) = x_1^2 + x_2^2 - 1 \) are the objective function and the standard nonlinear programming constraints for this problem. The function \( h(x,y) \) is given by:

\[
h(x,y) = \begin{bmatrix} x_1 - y_1 + y_2 - 1 \\ x_2 + y_2 \end{bmatrix}
\]

and

\[
a = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad b = \begin{bmatrix} \infty \\ 1 \end{bmatrix}.
\]

This example is written very succinctly in GAMS notation as:

```gams
$TITLE simple mpec example

variable f, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up = 1;
equations cost, g, h1, h2;

cost.. f =E= x1 + x2;
g.. sqr(x1) + sqr(x2) =L= 1;
h1.. x1 =G= y1 - y2 + 1;
h2.. x2 + y2 =N= 0;

model example / cost, g, h1, h2 /;
solve example using mpec min f;
```

Note that the equation `cost` is used to define \( f \), the constraint `g` defines the function \( g \), and \( h \) is defined by \( h1 \) and \( h2 \). The complementarity constraints utilize the standard GAMS convention of specifying the orthogonality relationship between \( h \) and \( y \) in the `model` statement. The interpretation of the `.` relies on the bounds \( a \) and \( b \) that are specified using `positive`, `negative`, or `lo` and `up` keywords in GAMS. Note that since \( h2 \) really specifies a function \( h2 \) and not a constraint \( h2(x,y) = 0 \), we use the GAMS syntax `=N=` to ensure this is clear here. Since the relationships satisfied by \( h1 \) and \( h2 \) are determined by the bounds, `=N=` could also be replaced by `=N=` in \( h1 \).

In describing the various reformulations for (6), it is convenient to partition the \( y \) variables into free \( \mathcal{F} \), lower bounded \( \mathcal{L} \), upper bounded \( \mathcal{U} \) and doubly bounded \( \mathcal{B} \) variables respectively, that is:

\[
\mathcal{B} := \{ y = (y_a, y_{\mathcal{F}}, y_{\mathcal{U}}, y_{\mathcal{B}}) : a \leq y_a \leq b, y_{\mathcal{F}} \leq y_{\mathcal{U}} \leq y_{\mathcal{B}} \}.
\]

We will assume (without loss of generality) that \( a_b < b_b \). If \( a_i = b_i \) then (6) holds trivially for the index \( i \) and we can remove the constraint \( h_i \) and its corresponding (fixed) variable \( y_i \) from the model. The complementarity condition for variables in \( y_i \in \mathcal{F} \) is simply the equality \( h_i(x,y) = 0 \) so these equality constraints are moved directly into the NLP constraints \( g \) of the original model as equalities. Thus, NLPEC needs only to treat the singly-bounded variables in \( \mathcal{L} \) and \( \mathcal{U} \) and the doubly-bounded variables in \( \mathcal{B} \). In the above example, \( \mathcal{F} = \{ 1 \} \), \( \mathcal{U} = \emptyset \) and \( \mathcal{B} = \{ 2 \} \).

### 3.1 Product reformulations

Product reformulations all involve products of \( y \) with \( h_i \), or products of \( y \) with some auxiliary or slack variables that are set equal to \( h_i \). The underlying point is that the constraints (3) are entirely equivalent to the following system of equalities and inequalities:

\[
\begin{align*}
w_{\mathcal{F}} &= h_{\mathcal{F}}(x,y), \quad a_{\mathcal{F}} \leq y_{\mathcal{F}}, \quad w_{\mathcal{F}} \geq 0 \quad \text{and} \quad (y_{\mathcal{F}} - a_{\mathcal{F}})^T w_{\mathcal{F}} = 0 \\
w_{\mathcal{U}} &= h_{\mathcal{U}}(x,y), \quad y_{\mathcal{U}} \leq b_{\mathcal{U}}, \quad v_{\mathcal{U}} \geq 0 \quad \text{and} \quad (b_{\mathcal{U}} - y_{\mathcal{U}})^T v_{\mathcal{U}} = 0 \\
w_{\mathcal{B}} - v_{\mathcal{B}} &= h_{\mathcal{B}}(x,y), \quad a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}}, \quad w_{\mathcal{B}} \geq 0, \quad v_{\mathcal{B}} \geq 0 \\
(y_{\mathcal{B}} - a_{\mathcal{B}})^T w_{\mathcal{B}} &= 0, \quad (b_{\mathcal{B}} - y_{\mathcal{B}})^T v_{\mathcal{B}} &= 0.
\end{align*}
\]

Note that each inner product is a summation of products of nonnegative terms: a slack variable and the difference between a variable and its bound. In each of these products, either the slack variable or its complement must be zero in order to have a
solution. Complementarity is forced by the multiplication of these two terms. The above reformulation is specified using option reftype mult.

There are a number of variations on this theme, all of which can be specified via an options file. All of the inner products could be put into the same equation, left as in (7) above, or broken out into individual products (one for each \( i \in L \cup U \), two for each \( i \in B \)). For example, the complementarity constraints associated with lower bounded variables involve nonnegativity of \( w_i^L \), \( y_i^L \geq a_i^L \) and either of the following alternatives:

\[
(y_i^L - a_i^L)^T w_i^L = \sum_{i \in L} (y_i - a_i) w_i = 0
\]

or

\[
(y_i - a_i) w_i = 0, \ i = 1, \ldots, m
\]

These different levels of aggregation are chosen using option aggregate none|partial|full.

Since all of the inner products in (7) involve nonnegative terms, we can set the inner products equal to zero or set them \( \leq 0 \) without changing the feasible set. To choose one or the other, use the option constraint equality|inequality.

As a concrete example, consider the option file

reftype mult
aggregate none
constraint inequality

applied to the simple example given above. Such an option file generates the nonlinear programming model:

\[
\begin{align*}
\min_{x_1, x_2, y_1, y_2, w_1, w_2, v_2} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\
& \quad w_1 y_1 \leq \mu \\
& \quad w_2 - v_2 = x_2 + y_2, w_2 \geq 0, y_2 \in [-1, 1] \quad (y_2 + 1) w_2 \leq \mu, (1 - y_2) v_2 \leq \mu
\end{align*}
\]

(8)

By default, a single model is generated with the value \( \mu \) set to $0$. There are many examples (e.g. interior point codes, many LP and NLP packages, published results on reformulation approaches to MPEC) that illustrate the value of starting with a "nearly-complementary" solution and pushing the complementarity gap down to zero. For this reason, the inner products in (7) above are always set equal to (or \( \leq \)) a scalar \( \mu \) instead of zero. By default \( \mu \) is zero, but options exist to start \( \mu \) at a positive value (e.g. InitMu 1e-2), to decrease it by a constant factor in a series of looped solves (e.g. NumSolves 4, UpdateFac 0.1), and to solve one last time with a final value for \( \mu \) (e.g. FinalMu 0). If the following lines are added to the option file

initmu 1.0
num solves 4

then five consecutive solves of the nonlinear program (8) are performed, the first one using \( \mu = 1 \) and each subsequent solve dividing \( \mu \) by 10 (and starting the NLP solver at the solution of the previous model in this sequence).

As a final example, we use a combination of these options to generate a sequence of nonlinear programs whose solutions attempt to trace out the "central path" favored by interior point and barrier algorithms:

reftype mult
constraint equality
initmu 1.0
num solves 4
updatefac 0.1
finalmu 1e-6
produces 6 nonlinear programs of the form

$$\begin{align*}
\min_{x_1, x_2, y_1, y_2, w_1, w_2, v_2} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1, \quad w_1 \geq 0, \quad y_1 \geq 0 \\
& \quad w_1 y_1 = \mu \\
& \quad w_2 - v_2 = x_2 + y_2, \quad w_2, v_2 \geq 0, \quad y_2 \in [-1, 1], \quad (y_2 + 1)w_2 = \mu, \quad (y_2 - 1)v_2 = \mu
\end{align*}$$

for values of $\mu = 1, 0.1, 0.01, 0.001, 0.0001$ and $1e-6$.

**Slacks and doubly bounded variables**

Slack variables can be used to reduce the number of times a complex nonlinear expression appears in the nonlinear programming model, as was carried out in (7). For a simpler illustrative example the NCP constraints (5) are equivalent to the constraints:

$$w_i = h_i(x, y), 0 \leq w_i, 0 \leq y_i \text{ and } y_i w_i = 0, i = 1, \ldots, m$$

This reformulation has an additional equality constraint, and additional variables $w$, but the expression $h_i$ only appears once. There are cases when this formulation will be preferable, and the simple option `slack none|positive` controls the use of the $w$ variables.

When there are doubly bounded variables present, these two slack options work slightly differently. For the positive case, the reformulation introduces two nonnegative variables $w_i$ and $v_i$ that take on the positive and negative parts of $h_i$ at the solution as shown in (7). Since this is the default value of the option `slack`, the example (8) shows what ensues to both singly and doubly bounded variables under this setting.

For the case `slack none`, Scholtes proposed a way to use a multiplication to force complementarity that requires no slack variables:

$$h_i \perp a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i, \ (y_i - a_i)h_i \leq \mu, \ (y_i - b_i)h_i \leq \mu \quad (9)$$

Note that unlike the inner products in Section Reformulation, we can expect that one of the inequalities in (9) is unlikely to be binding at a solution (i.e. when $h_i$ is nonzero). Therefore, we cannot use an equality in this reformulation, and furthermore the products must not be aggregated. Thus, if you use this option, the reformulation automatically enforces the additional options `constraint inequality` and `aggregate none` on the doubly bounded variables, even if the user specifies a conflicting option. Thus the option file

```
reftype mult
slack none
```

results in the model

$$\begin{align*}
\min_{x_1, x_2, y_1, y_2} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad x_1 - y_1 + y_2 - 1 \geq 0, \quad y_1 \geq 0 \\
& \quad (x_1 - y_1 + y_2 - 1)y_1 = \mu \\
& \quad y_2 \in [-1, 1], \quad (y_2 + 1)(x_2 + y_2) \leq \mu, \quad (y_2 - 1)(x_2 + y_2) \leq \mu
\end{align*}$$

Note that the complementarity constraint associated with $y_1$ is an equality (the default) while the constraints associated with $y_2$ are inequalities for the reasons outlined above.

In the case of doubly bounded variables, a third option is available for the slack variables, namely `slack one`. In this case, only one slack is introduced, and this slack removes the need to write the function $h_i$ twice in the reformulated model as follows:

$$h_i(x, y) \perp a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i, \ w_i = h_i(x, y), \ (y_i - a_i)w_i \leq \mu, \ (y_i - b_i)w_i \leq \mu$$

Note that the slack variable $Sw$ that is introduced is a free variable. It is not known before solving the problem whether $w_i$ will be positive or negative at the solution.

We take this opportunity to introduce a simple extension to our option mechanism, namely the ability to set the options for singly and doubly bounded variables differently. For example, the option file
sets the option slack positive for the singly bounded variables and the option slack one for the doubly bounded variables resulting in the model

\[
\begin{align*}
\min_{x_1,x_2,y_1,y_2,w_1,w_2} & \quad x_1 + x_2 \\
\text{subject to} & \quad x_1^2 + x_2^2 \leq 1 \\
& \quad w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\
& \quad w_1 y_1 = \mu_1 \\
& \quad w_2 = x_2 + y_2, y_2 \in [-1,1], (y_2 + 1)w_2 \leq \mu_2, (y_2 - 1)w_2 \leq \mu_2
\end{align*}
\]

Additional options such as

- `initmu 1.0 3.0`
- `num solves 2`
- `updatefac 0.1 0.2`

allow the values of \( \mu \) for the singly and doubly bounded variables to be controlled separately. In this case \( \mu_1 \) takes on values of 1, 0.1 and 0.01, while \( \mu_2 \) takes on values 3.0, 0.6 and 0.12 in each of the three nonlinear programming models generated.

### 3.2 NCP functions

An NCP-function is a function \( \phi(r,s) \) with the following property:

\[
\phi(r,s) = 0 \iff r \geq 0, s \geq 0, rs = 0
\]

Clearly, finding a zero of an NCP-function solves a complementarity problem in \((r,s)\). We can replace the inner products of nonnegative vectors in (7) with a vector of NCP functions whose arguments are complementary pairs, e.g. \((y'_\mathcal{L} - a'_\mathcal{L})^T w_{\mathcal{L}} = 0\) becomes \(\phi(y_i - a_i, w_i) = 0, i \in \mathcal{L}\) and arrive at another way \(\text{to treat the complementarity conditions.}\) Note that an NCP function forces both nonnegativity and complementarity, so constraints to explicitly force nonnegativity are not required, though they can be included.

Examples of NCP functions include the min function, \(\min(r,s)\), and the Fischer-Burmeister function

\[
\phi(r,s) = \sqrt{r^2 + s^2} - r - s
\]

There is no requirement that an NCP function be nonnegative everywhere (it may be strictly negative at some points), so there is little point in setting the option constraint; it will automatically take on the value constraint equality. NCP functions cannot be aggregated, so the aggregate option will always be set to none.

Since the arguments to the NCP functions are going to be nonnegative at solution, we cannot use the functions \(h_i\) directly in the case of doubly-bounded variables. We must use slacks \(w - v = h_i\) to separate \(Sh_iS\) into its positive and negative parts (but see Section Doubly bounded variables below). The slacks can be positive or free, since the NCP function will force positivity at solution. For the singly-bounded variables, slacks are optional, and can also be positive or free.

Both of the NCP functions mentioned above suffer from being non-differentiable at the origin (and at points where \(r = s\) for the min function). Various smoothed NCP-functions have been proposed that are differentiable. These smooth functions are parameterized by \(\mu\), and approach the true NCP-function as the smoothing parameter approaches zero. For example, the Fischer-Burmeister function includes a perturbation \(\mu\) that guarantees differentiability:

\[
\phi_{FB}(r,s) := \sqrt{r^2 + s^2} + 2\mu - (r + s)
\]

You can choose these particular NCP functions using option `RefType minFB|fFB`. The difference between the last two is that `RefType FB` writes out GAMS code to compute the function \(\phi_{FB}\), while `RefType fFB` makes use of a GAMS intrinsic function `NCPFB(r,s,\mu)` that computes \(\phi_{FB}\) internally. In general, using the GAMS intrinsic function should work better since the intrinsic can guard against overflow, scale the arguments before computing the function, and use alternative formulas that give more accurate results for certain input ranges.

As an example, the option file
A smoothed version of the min function was proposed by Chen & Mangasarian: 

\[ \mu \]

with a value of \( \mu = 0.01 \). Thus, the option file

\texttt{reftype fFB}

\texttt{slack free}

\texttt{initmu 1e-2}

generates the reformulation

\[
\begin{array}{ll}
\min & x_1 + x_2 \\
\text{subject to} & x_1^2 + x_2^2 \leq 1 \\
& w_1 = x_1 - y_1 + y_2 - 1 \\
& \phi_{FB}(w_1, y_1, \mu) = 0 \\
& w_2 - y_2 = x_2 + y_2 \\
& \phi_{FB}(y_2 + 1, w_2, \mu) = 0, \phi_{FB}(1 - y_2, v_2, \mu) = 0
\end{array}
\]

with a value of \( \mu = 0.01 \). Following a path of solutions for decreasing values of \( \mu \) is possible using the options discussed above.

Each of the two arguments to the NCP function will be nonnegative at solution, but for each argument we have the option of including a nonnegativity constraint explicitly as well. This results in the 4 values for the option NCPBounds none/all/function/variable. When no slacks are present, this option controls whether to bound the function \( h_i \) as well as including it in the NCP function, e.g. \( h_i \geq 0, \phi(h_i, y_i - a_i) = 0 \). When slacks are present, we require that the slack setting be consistent with the bound setting for the function argument to the NCP function, where NCPBounds none/variable is consistent with free slack variables and NCPBounds all/function is consistent with positive slack variables.

Thus, the option file

\texttt{reftype min}

\texttt{slack positive}

\texttt{NCPBounds function}

generates the reformulation

\[
\begin{array}{ll}
\min & x_1 + x_2 \\
\text{subject to} & x_1^2 + x_2^2 \leq 1 \\
& w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0 \\
& \min(w_1, y_1) = \mu \\
& w_2 - y_2 = x_2 + y_2, w_2, v_2 \geq 0 \\
& \min(y_2 + 1, w_2) = \mu, \min(1 - y_2, v_2) = \mu
\end{array}
\]

The NCPBounds function option means that the variable argument to the NCP function (in this case \( y \)) does not have its bounds explicitly enforced. It should be noted that this nonlinear program has nondifferentiable constraints for every value of \( \mu \). For this reason, the model is constructed as a \texttt{dnlp} model (instead of an \texttt{nlp} model) in GAMS.

A smoothed version of the min function was proposed by Chen & Mangasarian:

\[
\phi_{CM}(r, s) := r - \mu \log(1 + \exp((r - s)/\mu)). \tag{11}
\]

This function is not symmetric in its two arguments, so \( \phi_{CM}(r, s) \neq \phi_{CM}(s, r) \). For this reason, we distinguish between the two cases. Unlike the Fischer-Burmeister function \( \phi_{FB} \), \( \phi_{CM} \) is not defined in the limit (i.e. for \( \mu = 0 \)) if you use GAMS code to compute it. However, the GAMS intrinsic NCPCRM(r,s,mu) handles this limit case internally. The option \texttt{RefType CMfx|CMfx|CMfx|CMfx} chooses a reformulation based on the function \( \phi_{CM} \). Again, the last two choices use the GAMS intrinsic function.

**Doubly bounded variables**

Like the mult formulation (7), reformulations using NCP functions are appropriate as long as we split the function \( h_i \) matching a doubly-bounded variable into its positive and negative parts \( w_i - v_i = h_i \). To avoid this, Billups has proposed using a composition of NCP functions to treat the doubly-bounded case:

\[
h_i \quad a_i \leq y_i \leq b_i \iff \phi_{FB}(y_i - a_i, \phi_{FB}(b_i - y_i, -h_i)) = 0 \tag{12}
\]
Use option RefType Bill|fBill to choose such a reformulation for the doubly-bounded variables. The first option value writes out the function in explicit GAMS code, while the second writes it out using the GAMS intrinsic function NCPFB.

### 3.3 Penalty functions

All of the reformulations discussed so far have reformulated the complementarity conditions as constraints. It is also possible to write out the function in explicit GAMS code, while the second writes it out using the GAMS intrinsic function NCPFB.

Choose this treatment using option refType penalty. The options aggregate and constraint are ignored, since the inner products here are all aggregated and there are no relevant constraints. It is possible to do a similar reformulation without using slacks, so the options slack none|positive can be used in conjunction with this reformulation type.

The following option file shows the use of the penalty reformulation, but also indicates how to use a different reformulation for the singly and doubly bounded variables:

```plaintext
reftype penalty mult
slack none *
initmu 1.0
numsolves 2
updatefac 0.1 0.2
```

applied to the simple example given above. The "*" value allows the slack option to take on its existing value, in this case positive. Such an option file generates the nonlinear programming model:

\[
\min_{x,y,w,v} \quad x_1 + x_2 + \frac{1}{\mu_1} y_1 (x_1 - y_1 + y_2 - 1) \\
\text{subject to} \quad x_1^2 + x_2^2 \leq 1 \\
\quad x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \]

\[
w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1] \\
\quad (y_2 + 1)w_2 \leq \mu_2, (1 - y_2)v_2 \leq \mu_2
\]

The penalty parameter \(\mu_1\) is controlled separately from the doubly bounded constraint parameter \(\mu_2\). For consistency with other options, the penalty parameter in the objective is \(1/\mu\) meaning that as \(\mu_1\) tends to zero, the penalty increases. The option initmu has only one value, so both the singly and doubly bounded \(\mu\) values are initialized to 1. In the above example, three solves are performed with \(\mu_1 = 1.0.1\) and 0.01 and \(\mu_2 = 1.0.2\) and 0.04.

### 3.4 Testing for complementarity

In some cases a solution to the reformulated model may not satisfy the complementarity constraints of the original MPEC, e.g. if a large penalty parameter is used in the reformulation. It can also happen that the solution tolerances used in the NLP solver allow solutions with small error in the NLP model but large error in the original MPEC. For example if \(x = f(x) = .001\) then the NLP constraint \(xf(x) = 0\) may satisfy the NLP feasibility tolerance but it’s not so easy to claim that either \(x\) or \(f(x)\) is zero. The NLPEC solver includes a check that the proposed solution does in fact satisfy the complementarity constraints. The complementarity gap is computed using the definition common to all GAMS MCP solvers in computing the objval model attribute for an MCP model. The tolerance used for this complementarity gap can be adjusted using the testtol option.
4 Options

For details on how to create and use an option file, see the introductory chapter on solver usage.

For most GAMS solvers, the use of an options file is discouraged, at least for those unfamiliar with the solver. For NLPEC, however, we expect that most users will want to use an options file from the very beginning. NLPEC is as much a tool for experimentation as it is a solver, and as such use of the options file is encouraged.

Option values can take many different types (e.g. strings, integers, or reals). Perhaps the most important option to remember is one with no value at all: the help option. Help prints a list of the available options, along with their possible values and some helpful text. The options file is read sequentially, so in case an option value is set twice, the latter value takes precedence. However, any consistency checks performed on the options values (e.g. RefType fBill cannot be used with aggregate full) are made after the entire options file is read in, so the order in which different options appear is not important, provided the options are not specified twice.

4.1 Setting the Reformulation Options

While NLPEC has many options, there is a small set of five options that, taken together, serve to define the type of reformulation used. Listed in order of importance (highest priority items first), these reformulation options are the RefType, slack, constraint, aggregate and NCPBounds options. In some cases, setting the highest-priority option RefType is enough to completely define a reformulation (e.g. RefType penalty in the case of doubly-bounded variables). In most cases though, the lower-priority options play a role in defining or modifying a reformulation. It’s useful to consider the reformulation options in priority order when creating option files to define reformulations.

Some of the combinations of the reformulation options don’t make sense. For example, the use of an NCP function to force complementarity between its two input arguments requires a separate function for each complementary pair, so setting both RefType min and aggregate full is inconsistent. NLPEC implements consistency checks on the reformulation options using the priority order: Given a consistent setting of the higher priority options, the next-highest priority option is checked and, if necessary, reset to be consistent with the items of higher priority. The end result is a set of consistent options that will result in a working reformulation. NLPEC prints out the pre- and post-checked sets of reformulation options, as well as warning messages about changes made. In case you want to use an option that NLPEC doesn’t think is consistent, you can use the NoCheck option: this supresses the consistency checks.

Each of the reformulation options in the table below takes two values - one for the singly-bounded variables in \( L \cup U \) and another for the doubly-bounded variables in \( B \). If one option value appears, it sets both option values. When setting both option values, use an asterisk “∗” to indicate no change. So for example, an option file

4.2 Reformulation Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate</td>
<td>controls constraint aggregation \nDetermines if certain constraints are aggregated or not. E.g. to force ( w \geq 0 ) and ( y \geq 0 ) to be complementary we can write either ( w^T y \leq 0 ) or ( w_i^T y_i = 0 ), for all ( i ). \nnone use no aggregation \npartial aggregate doubly-bounded variables separately from others \nfull use maximum aggregation possible</td>
<td>none</td>
</tr>
<tr>
<td>constraint</td>
<td>controls use of equality/inequality \nDetermines if certain constraints are written down using equalities or inequalities. E.g. to force ( w \geq 0 ) and ( y \geq 0 ) to be complementary we can write either ( w^T y \leq 0 ) or ( w^T y = 0 ). This option only plays a role when bounding a quantity whose sign cannot be both positive and negative and which must be 0 at a solution. \nequality use =E= constraints \ninequality use =G= constraints</td>
<td>equality</td>
</tr>
</tbody>
</table>
NCPBounds | sets explicit bounds on arguments of NCP functions
---|---
| Determines which of the two arguments to an NCP function \( \Phi(r,s) \) are explicitly constrained to be nonnegative. The explicit constraints are in addition to those imposed by the constraint \( \Phi(r,s) = 0 \), which implies nonnegativity of \( r \) and \( s \).
| none no explicit bounds included
| function explicit bounds on function/slacks included
| variable explicit bounds on variables included
| all both function and variable bounds included

refType | reformulation type
---|---
| Determines the type of reformulation used. Our notation and descriptions are taken from a special case of the MPEC, the NCP: find \( x > 0, f(x) >= 0, x^T f(x) = 0 \).
| mult inner product \( <x,f> = 0 \)
| min NCP function min\((x,f)\)
| CMxf Chen-Mangasarian NCP function, explicit
| CMfx Chen-Mangasarian NCP function, explicit
| fCMxf Chen-Mangasarian NCP function, intrinsic
| fCMfx Chen-Mangasarian NCP function, intrinsic
| FB Fischer-Burmeister NCP function, explicit
| fFB Fischer-Burmeister NCP function, intrinsic
| FB_neg Fischer-Burmeister NCP function negated, explicit
| fFB_neg Fischer-Burmeister NCP function negated, intrinsic
| Bill Billups function for doubly-bounded variables, explicit
| fBill Billups function for doubly-bounded variables, intrinsic
| penalty objective penalty
| median something else
| fVUsin Veelken-Ulbrich NCP function (smoothed min), intrinsic
| fVUpow Veelken-Ulbrich NCP function (smoothed min), intrinsic

slack | what type of slacks to put in
---|---
| Determines if slacks are used to treat the functions \( h_{i} \). For single-bounded variables, we use at most one slack (either free or positive) for each \( h_{i} \). For doubly-bounded variables, we can have no slacks, one slack (necessarily free), or two slacks (either free or positive) for each \( h_{i} \).
| none no slacks will be used
| free free slacks will be used
| positive non-negative slacks will be used
| one one free slack will be used for each \( h_{i} \) in the doubly bounded case

### 4.3 General Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>allSolves</td>
<td>do all solves in a loop regardless of previous failure&lt;br&gt;In case multiple (looped) solves are specified, the default is to skip subsequent solves when any solve terminates without getting a solution. Setting this flag removes the check and all solves are done, regardless of previous failures.</td>
<td>0</td>
</tr>
<tr>
<td>dotGams</td>
<td>name of gams source file for scalar model</td>
<td>nlpec.gms</td>
</tr>
<tr>
<td>dumpValid</td>
<td>dump valid reformulation options to a GDX file and exit</td>
<td>0</td>
</tr>
<tr>
<td>equReform</td>
<td>outdated and deprecated</td>
<td>0</td>
</tr>
<tr>
<td>finalMu</td>
<td>final value of parameter ( \mu )&lt;br&gt;If specified, an extra solve is carried out with ( \mu ) set to this value. Can be set independently for singly and doubly bounded variables.</td>
<td>0</td>
</tr>
</tbody>
</table>
initMu | initial value of parameter mu  
A single solve of the nonlinear program is carried out for this value. Note that \( \mu \) must be positive for some settings of \texttt{RefType}, e.g. penalty. Can be set independently for singly and doubly bounded variables. | 0  

initSLo | lower bound for artificials added to the problem | 0  

initSUp | upper bound for artificials added to the problem | \texttt{maxdouble}  

noCheck | do not check consistency of reformulation options | 0  

numSolves | number of extra solves  
This should be set in conjunction with the \texttt{updateFac} option. | 0  

subSolver | subsolver to run |  

subSolverOpt | optfile value to pass to the subsolver | 0  

terminate | terminate after generating scalar GAMS source code | 0  

testTol | tolerance for complementarity check in MPEC/MCP | 1e-005  

updateFac | update factor for \( \mu \)  
The factor that multiplies \( \mu \) before each of the extra solves triggered by the \texttt{numSolves} option. Can be set independently for singly and doubly bounded variables. | 1e-1  

### 4.4 The Outdated \texttt{equreform} Option

In the early versions of NLPEC the only way to set the reform type was via the \texttt{equreform} option. Each valid \texttt{equreform} value represented a preselected combination of the options from Section Setting the Reformulation Options. This made it difficult to experiment with combinations not preselected, so the options in Section Setting the Reformulation Options were added. Be default, the \texttt{equreform} option has value 0 and is not used. To get the old behavior, set \texttt{equreform} to a positive value - this will force the options in Section Setting the Reformulation Options to be ignored. The general options in Section General Options are used no matter how the reformulation type is selected - via \texttt{RefType} or \texttt{equreform}.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{equreform}</td>
<td>Old way to set the type of reformulation used.</td>
<td>0</td>
</tr>
</tbody>
</table>

The values allowed for \texttt{equreform} and their implications are given in Table 1.

### 5 Open Architecture

In this section we describe the architecture of the NLPEC solver, i.e. the way the solver is put together. This should be useful to anybody using NLPEC for experiments or to those wanting to know the details of how NLPEC works.

The foundation for the NLPEC solver is the software library (also used in the GAMS/CONVERT solver) that allows us to write out a scalar GAMS model that is mathematically equivalent to the original, or to write out selected pieces of such a model. Using this software, NLPEC creates a GAMS NLP model (default name: \texttt{nlpec.gms}) using one of the reformulation strategies from Section Reformulation. This model may contain many new variables and/or equations, but it will surely contain the (non)linear expressions defining the original model as well. Once the model \texttt{nlpec.gms} has been created, NLPEC calls gams to solve this model, using the current NLP solver. After the model has solved, NLPEC reads the NLP solution, extracts the MPEC solution from it, and passes this MPEC solution back to GAMS as it terminates.

There are a number of advantages to this architecture. First, its openness makes it easy to see exactly what reformulation is being done. The intermediate NLP file \texttt{nlpec.gms} is always available after the run for those wishing to know the details about the reformulation or for debugging in case things didn’t work out as expected. It would also be possible to modify this file to do some quick and dirty experiments with similar reformulation strategies. Another advantage is the variety of NLP solvers that can be plugged in to solve the reformulated model. There is no need to program (and debug) an interface to an NLP package to run experiments with an NLP solver - the existing GAMS link is all that is needed. It is also easy to experiment with non-default solver options that may be more appropriate for reformulated MPEC models or for a particular choice of
<table>
<thead>
<tr>
<th>equref</th>
<th>reftype</th>
<th>sign</th>
<th>slacks</th>
<th>free-y</th>
<th>reftype</th>
<th>sign</th>
<th>slacks</th>
<th>free-y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$&lt;,&gt;_{i}$</td>
<td>= $\mu$</td>
<td>bnd</td>
<td>$&lt;,&gt;_{i}$</td>
<td>= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$&lt;,&gt;_{i}$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td>$&lt;,&gt;_{i}$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$&lt;,&gt;_{i}$</td>
<td>= $\mu$</td>
<td>bnd</td>
<td>Scholtes</td>
<td>&lt;= $\mu$</td>
<td>one</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$&lt;,&gt;_L+U+B$</td>
<td>= $\mu$</td>
<td>bnd</td>
<td>$&lt;,&gt;_L+U+B$</td>
<td>= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$&lt;,&gt;_L+U+B$</td>
<td>= $\mu$</td>
<td>none</td>
<td>$&lt;,&gt;_L+U+B$</td>
<td>= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$&lt;,&gt;_L+U$</td>
<td>= $\mu$</td>
<td>none</td>
<td>Scholtes</td>
<td>&lt;= $\mu$</td>
<td>one</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$&lt;,&gt;_L+U$</td>
<td>&lt;= $\mu$</td>
<td>none</td>
<td>Scholtes</td>
<td>&lt;= $\mu$</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$&lt;,&gt;_{i}$</td>
<td>= $\mu$</td>
<td>none</td>
<td>Scholtes</td>
<td>&lt;= $\mu$</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$&lt;,&gt;_obj$</td>
<td>bnd</td>
<td>$&lt;,&gt;_obj$</td>
<td>bnd</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$&lt;,&gt;_obj$</td>
<td>none</td>
<td>$&lt;,&gt;_obj$</td>
<td>bnd</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$&lt;,&gt;_{i}$</td>
<td>= $\mu$</td>
<td>none</td>
<td>$&lt;,&gt;_{i}$</td>
<td>= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>F-B</td>
<td>= 0</td>
<td>none</td>
<td>F-B</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>F-B</td>
<td>= 0</td>
<td>none</td>
<td>F-B</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>min</td>
<td>= $\mu$</td>
<td>free</td>
<td>min</td>
<td>= $\mu$</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>min</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td>min</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>C-M($x, f$)</td>
<td>= 0</td>
<td>free</td>
<td>C-M($x, f$)</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>C-M($x, f$)</td>
<td>= 0</td>
<td>bnd</td>
<td>C-M($x, f$)</td>
<td>= 0</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>$&lt;,&gt;_L+U$</td>
<td>= $\mu$</td>
<td>none</td>
<td>$&lt;,&gt;_B$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>$&lt;,&gt;_{i}$</td>
<td>&lt;= $\mu$</td>
<td>none</td>
<td>$&lt;,&gt;_{i}$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>$&lt;,&gt;_L+U$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td>$&lt;,&gt;_B$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>$&lt;,&gt;_L+U+B$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td>$&lt;,&gt;_L+U+B$</td>
<td>&lt;= $\mu$</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>F-B</td>
<td>= 0</td>
<td>bnd</td>
<td>F-B</td>
<td>= 0</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>F-B</td>
<td>= 0</td>
<td>free</td>
<td>F-B</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>C-M($f, x$)</td>
<td>= 0</td>
<td>none</td>
<td>C-M($f, x$)</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>C-M($f, x$)</td>
<td>= 0</td>
<td>none</td>
<td>C-M($f, x$)</td>
<td>= 0</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>NCPF</td>
<td>= 0</td>
<td>none</td>
<td>NCPF</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>NCPF</td>
<td>= 0</td>
<td>none</td>
<td>NCPF</td>
<td>= 0</td>
<td>bd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>NCPF</td>
<td>= 0</td>
<td>bnd</td>
<td>NCPF</td>
<td>= 0</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>NCPF</td>
<td>= 0</td>
<td>free</td>
<td>NCPF</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>NCPCM($x, f$)</td>
<td>= 0</td>
<td>free</td>
<td>NCPCM($x, f$)</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>NCPCM($x, f$)</td>
<td>= 0</td>
<td>bnd</td>
<td>C-M($x, f$)</td>
<td>= 0</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>NCPCM($f, x$)</td>
<td>= 0</td>
<td>none</td>
<td>NCPCM($f, x$)</td>
<td>= 0</td>
<td>free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>NCPCM($f, x$)</td>
<td>= 0</td>
<td>none</td>
<td>NCPCM($f, x$)</td>
<td>= 0</td>
<td>bnd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 44.4: Values for equreform option.
reformulation.
Optimization Services (OS)

COIN-OR OS (Optimization Services) is an initiative to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. The code has been written primarily by Horand Gassmann, Jun Ma, and Kipp Martin. Kipp Martin is the COIN-OR project leader for OS.

For more information we refer to the web sites http://www.optimizationservices.org and https://projects.coin-or.org/OS, the OS manual [4], and the papers [1], [2], [3].

With the OS link in GAMS, you can send your instance to an Optimization Services Server for remote solving. You must have access to such a server to use this tool.

OS supports continuous, binary, integer, semicontinuous, and semiinteger variables and linear and nonlinear equations. Special ordered sets and indicator constraints are currently not supported. Initial values are currently not supported by the GAMS/OS link.

1  Usage

The following statement can be used inside your GAMS program to specify using OS

    Option MINLP = OS;  { or LP, RMIP, MIP, DNLP, NLP, RMINLP, QCP, RMIQCP, MIQCP }

The above statement should appear before the Solve statement.

By default, for a given instance of a GAMS model, nothing happens. To solve an instance remotely, you have to specify the URL of an Optimization Services Server via the option service. Usually, the server chooses an appropriate solver for your instance, depending on their availability on the server. A fully equipped server chooses CLP for continuous linear models (LP and RMIP), IPOPT for continuous nonlinear models (NLP, DNLP, RMINLP, QCP, RMIQCP), CBC for mixed-integer linear models (MIP), and BONMIN for mixed-integer nonlinear models (MIQCP, MINLP). An easy way to influence the choice of the solver on the server is the solver option.

Further options can be provided in an OSoL (Optimization Services Options Language) file, which is specified via the the readosol option. An example OSoL file looks like

    <?xml version="1.0" encoding="UTF-8"?>
    <osol xmlns="os.optimizationservices.org" xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="os.optimizationservices.org http://www.optimizationservices.org/schemas/2.0/OSoL.xsd">
        <optimization>
          <solverOptions numberOfSolverOptions="3">
            <solverOption name="cuts" solver="cbc" value="off" />
            <solverOption name="max_active_nodes" solver="symphony" value="2" />
            <solverOption name="max_iter" solver="ipopt" type="integer" value="2000" />
          </solverOptions>
        </optimization>
    </osol>

It specifies that if CBC is used, then cutting planes are disabled, if SYMPHONY is used, then at most 2 nodes should be active, and if IPOPT is used, then a limit of 2000 iterations is imposed.
By default, the call to the server is a *synchronous* call. The GAMS process will wait for the result and then display the result. This may not be desirable when solving large optimization models. In order to use the remote solver service in an *asynchronous* fashion, one can make use of the GAMS Grid Computing Facility, see Appendix I in the GAMS manual.

## 2 Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>readosol</td>
<td>read the solver options from an OSoL file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifies the name of an option file in OSoL format that is given to the OS server. This way it is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>possible to pass options directly to the solvers interfaced by OS.</td>
<td></td>
</tr>
<tr>
<td>service</td>
<td>address of an Optimization Services Agent for a remote solve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifies the URL of an Optimization Services Server. The GAMS model is converted into OSiL format,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>send to the server, and the result translated back into GAMS format. Note that by default the server</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chooses a solver that is appropriate to the model type. You can change the solver with the solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>option.</td>
<td></td>
</tr>
<tr>
<td>solver</td>
<td>specification of a solver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifies the solver that is used to solve an instance on the OS server.</td>
<td></td>
</tr>
<tr>
<td>writeosil</td>
<td>write the problem instance as OSiL file</td>
<td>osil.xml</td>
</tr>
<tr>
<td></td>
<td>Specifies the name of a file in which the GAMS model instance should be written in OSiL format.</td>
<td></td>
</tr>
<tr>
<td>writeosrl</td>
<td>write the optimization result as OSrL file</td>
<td>osrl.xml</td>
</tr>
<tr>
<td></td>
<td>Specifies the name of a file in which the result of a solve process (solution, status, ...) should be</td>
<td></td>
</tr>
<tr>
<td></td>
<td>written in OSrL format.</td>
<td></td>
</tr>
</tbody>
</table>

## Bibliography


OSICPLEX, OSIGUROBI, OSIMOSEK, OSISOPLEX, OSIXPRESS

The "bare bone" solver links GAMS/OSICPLEX, GAMS/OSIGUROBI, GAMS/OSIMOSEK, GAMS/OSISOPLEX, and GAMS/OSIXPRESS allow users to solve their GAMS models with SOPLEX or a standalone license of CPLEX, GUROBI, MOSEK, or XPRESS. Since SOPLEX is distributed under the ZIB Academic License, it is only available for users with a GAMS academic license. The links use the COIN-OR Open Solver Interface (OSI) to communicate with these solvers. The OSICPLEX link has been written primarily by Tobias Achterberg, the OSIGUROBI link has been written primarily by Stefan Vigerske, the OSIMOSEK link has been written primarily by Bo Jensen, the OSISOPLEX link has been written primarily by Tobias Achterberg, Ambros M. Gleixner, and Wei Huang, and the OSIXPRESS link has been written primarily by John Doe. Matthew Saltzman is the COIN-OR project leader for OSI.

For more information we refer to the OSI web site https://projects.coin-or.org/Osi.

The OSI links support linear equations and continuous, binary, and integer variables. Semicontinuous and Semiinteger variables, special ordered sets, branching priorities, and indicator constraints are not supported by OSI. OSISOPLEX solves only LPs, no MIPs.

1 Usage

The following statement can be used inside your GAMS program to specify using OSIGUROBI

```
Option MIP = OSIGUROBI;    { or LP or RMIP }
```

The above statement should appear before the Solve statement.

The links support the general GAMS options reslim, optca, optcr, nodlim, iterlim, and threads (except for OSISOPLEX). For OSICPLEX, OSIGUROBI, OSIMOSEK, and OSIXPRESS an option file in the format required by the solver can be provided via the GAMS optfile option. See Section Option files for details.

If a MIP is solved via one of the OSI links, only primal solution values are reported by default. To receive also the dual values for the LP that is obtained from the MIP by fixing all discrete variables, the GAMS option integer1 must be set to a nonzero value. Note that this may lead to solving another LP after the MIP solve has finished.

Setting the GAMS option integer2 to a nonzero value makes variable and equation names available to the solver. This option may be useful for debugging purposes.

Setting the GAMS option integer3 to a nonzero value leads to writing the model instance to a file in LP or MPS format before starting the solution process (integer3=1 writes an MPS file, integer3=2 writes an LP files, integer3=4 writes a native MPS file; sum these values to write several files). The name of the MPS file is chosen to be the name of the GAMS model file with the extension .gms replaced by .mps. This option may be useful for debugging purposes.

For OSICPLEX, OSIGUROBI, and OSIXPRESS, setting the GAMS option integer4 to a nonzero value leads to passing the variable level values (.l suffix) to the MIP solver as initial solution. This is analogous to the mipstart option of the full CPLEX and GUROBI links and the loadmipsol option of the full XPRESS link.
1.1 Option files

**OSICPLEX Options**

In an OSICPLEX option file, each line lists one option setting, where the option name and value are separated by space. Example:

```plaintext
CPX_PARAM_MIPEMPHASIS 2
CPX_PARAM_HEURFREQ 42
CPX_PARAM_MIPDISPLAY 4
```

**OSIGUROBI Options**

In an OSIGUROBI option file, each line lists one option setting, where the option name and value are separated by space. Example:

```plaintext
Cuts 2
Heuristics 0.1
```

**OSIMOSEK Options**

An OSIMOSEK option file begins with the line BEGIN MOSEK and terminates with END MOSEK. Comments are introduced with an '%', empty lines are ignored. Each other line starts with a MOSEK parameter value, followed by space, and a value for that parameter. Example:

```plaintext
BEGIN MOSEK
  % disable probing and solve the root node by the interior point solver
  MSK_IPAR_MIO_PRESOLVE_PROBING MSK_OFF
  MSK_IPAR_MIO_ROOT_OPTIMIZER MSK_OPTIMIZER_INTPNT
END MOSEK
```

**OSIXPRESS Options**

In an OSIXPRESS option file, each line lists one option setting, where the option name and value are separated by an equal sign. Example:

```plaintext
MIPLOG = 3
HEURFREQ = 2
```
# SCIP

## Contents

1. Usage ................................................................. 894
   1.1 Specification of SCIP Options .................................. 895
   1.2 Specification of Indicators ..................................... 895
2. Special Features ...................................................... 896
   2.1 SCIP interactive shell ........................................ 896
   2.2 Emphasis Settings .............................................. 896
   2.3 Solution Pool .................................................. 897
   2.4 Solving process tracing ........................................ 897
   2.5 Notes on solving MINLPs with SCIP ............................ 897
3. List of SCIP Options .................................................. 897
   3.1 gams ............................................................ 898
   3.2 gams/solvetrace ................................................ 898
   3.3 branching ...................................................... 898
   3.4 branching/allfullstrong ....................................... 899
   3.5 branching/cloud ............................................... 899
   3.6 branching/fullstrong .......................................... 899
   3.7 branching/inference ........................................... 900
   3.8 branching/leastroinf ......................................... 900
   3.9 branching/mostinf ............................................. 900
   3.10 branching/pscost .............................................. 901
   3.11 branching/random ............................................. 901
   3.12 branching/relpscost .......................................... 901
   3.13 conflict ....................................................... 902
   3.14 conflict/bounddisjunction .................................... 903
   3.15 conflict/indicatorconflict ..................................... 904
   3.16 conflict/linear ................................................ 904
   3.17 conflict/logicor ............................................... 904
   3.18 conflict/setppc ............................................... 904
   3.19 constraints .................................................... 904
   3.20 constraints/SOS1 ............................................... 905
   3.21 constraints/SOS2 ............................................... 905
   3.22 constraints/abspower .......................................... 905
   3.23 constraints/and ............................................... 906
   3.24 constraints/bivariate ......................................... 907
   3.25 constraints/bounddisjunction .................................. 907
   3.26 constraints/indicator ......................................... 908
   3.27 constraints/integral .......................................... 909
   3.28 constraints/knapsack .......................................... 910
   3.29 constraints/linear ............................................ 911
   3.30 constraints/linear/upgrade .................................... 912
3.31 constraints/logicor ................................................................. 912
3.32 constraints/nonlinear ............................................................ 913
3.33 constraints/nonlinear/upgrade ................................................ 913
3.34 constraints/quadratic .......................................................... 914
3.35 constraints/quadratic/upgrade ............................................... 915
3.36 constraints/setpce ............................................................... 915
3.37 constraints/soc ................................................................. 916
3.38 constraints/varbound .......................................................... 917
3.39 display ............................................................................ 917
3.40 display/avgdualbound .......................................................... 918
3.41 display/conflicts ............................................................... 918
3.42 display/conss ................................................................. 918
3.43 display/curcols ............................................................... 918
3.44 display/curconss ............................................................. 918
3.45 display/curdualbound ......................................................... 918
3.46 display/currows .............................................................. 919
3.47 display/cutoffbound ............................................................ 919
3.48 display/cuts ................................................................. 919
3.49 display/depth ................................................................. 919
3.50 display/dualbound ............................................................. 919
3.51 display/estimate .............................................................. 919
3.52 display/feasST ................................................................. 919
3.53 display/gap ................................................................. 920
3.54 display/lpavgiterations ....................................................... 920
3.55 display/lpcond .............................................................. 920
3.56 display/lpiterations ........................................................ 920
3.57 display/lpobj ............................................................... 920
3.58 display/maxdepth ............................................................... 920
3.59 display/menumused ........................................................... 921
3.60 display/nexternbranchcands ................................................. 921
3.61 display/nfrac ............................................................... 921
3.62 display/nmodes ............................................................. 921
3.63 display/nodesleft ............................................................ 921
3.64 display/nsols ................................................................. 921
3.65 display/plungedepth .......................................................... 921
3.66 display/poolszie ............................................................ 922
3.67 display/primalbound ......................................................... 922
3.68 display/primalgap ............................................................ 922
3.69 display/pseudoobj ............................................................ 922
3.70 display/separounds ......................................................... 922
3.71 display/solfound ............................................................ 922
3.72 display/sols ................................................................. 922
3.73 display/strongbranches ..................................................... 923
3.74 display/time ................................................................. 923
3.75 display-vars ................................................................. 923
3.76 heuristics/actconsdiving .................................................... 923
3.77 heuristics/clique ........................................................... 924
3.78 heuristics/coefdiving ...................................................... 924
3.79 heuristics/crossover ......................................................... 925
3.80 heuristics/dins ............................................................. 926
3.81 heuristics/dualval ........................................................... 926
3.82 heuristics/feaspump ........................................................ 927
SCIP (Solving Constraint Integer Programs) is developed at the Zuse Institute Berlin (ZIB) in cooperation with TU Darmstadt, RWTH Aachen, and University of Erlangen-Nürnberg, see the SCIP webpage for the current list of developers.

Since SCIP is distributed under the ZIB Academic License, it is only available for users with a GAMS academic license.

SCIP is a framework for Constraint Integer Programming oriented towards the needs of Mathematical Programming experts who want to have total control of the solution process and access detailed information down to the guts of the solver. SCIP can also be used as a pure MIP or MINLP solver or as a framework for branch-cut-and-price. Within GAMS, the MIP and MINLP solving facilities of SCIP are available.

For more detailed information, we refer to [1–6, 8] and the SCIP web site.

GAMS/SCIP uses CPLEX, if licensed, and otherwise SOPLEX [9] as LP solver, the COIN-OR Interior Point Optimizer IPOPT [7] as nonlinear solver, and CppAD to compute derivatives of nonlinear functions.

SCIP supports continuous, binary, integer, semi-continuous, semi-integer variables, indicator constraints, special ordered sets, and branching priorities for discrete variables.

1 Usage

The following statement can be used inside your GAMS program to specify using SCIP

Option MIP = SCIP; { or QCP or NLP or MIQCP or MINLP or ... }
The above statement should appear before the Solve statement. If SCIP was specified as the default solver during GAMS installation, the above statement is not necessary.

GAMS/SCIP currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/SCIP with BCH, please consider to use a GAMS system of version $\leq 23.3$.

1.1 Specification of SCIP Options

GAMS/SCIP supports the GAMS parameters reslim, iterlim, nodlim, optca, optcr, and workspace. Further, the option threads can be used to control the number of threads used in the linear algebra routines of IPOPT, see The linear solver in IPOPT in the IPOPT manual for details, and for solving LPs (if CPLEX’ barrier solver is used).

Options can be specified by a SCIP options file. A SCIP options file consists of one option or comment per line. A pound sign (#) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs). Further, string values have to be enclosed in quotation marks.

A small example for a scip.opt file is:

```
propagating/probing/maxprerounds = 0
separating/maxrounds = 0
separating/maxroundsroot = 0
```

It causes GAMS/SCIP to disable probing during presolve and to turn off all cut generators.

1.2 Specification of Indicators

Indicators are a modeling tool to specify that certain equations in a model must only be satisfied if certain binary variables take a specified value. Indicators are not supported by the GAMS language, but can be passed to SCIP via a separate file. The name of that file is specified via the option gams/indicatorfile in a SCIP option file.

The indicator specification file declares for some equations, for which value of which binary variables the equation is switched on. The syntax is

```
indic equation$variable onval
```

where equation is the name of the equation, variable is the name of the binary variable, and onval is either 0 or 1. The line specifies that equation has to hold whenever variable variable takes value onval.

For example, assume a GAMS model contains a set of equations of the form

```
equ1(i,j,k)$($ord(i)<ord(j)$)... lhs =l= rhs;
```

To specify that they only have to be satisfied if a binary variable

```
bin1(i,k)
```

takes the value 1, the indicator specification file should contain the line

```
indic equ1(i,j,k)$bin1(i,k) 1
```

More documentation can be found at http://www.gams.com/solvers/cpxindic.htm. In difference to the GAMS/C-PLEX interface, the indicator specifications need to be in a separate file for SCIP.

Currently, indicators can only be used for linear equations.
2 Special Features

2.1 SCIP interactive shell

The interactive shell in SCIP is a powerful tool that allows the user to display various information (e.g., branching statistics, presolved model), load emphasis settings, interrupt a solve to change parameters or trigger a restart, write the model in various file formats, start SCIPs solution counter, and many more things.

When setting the option `gams/interactive` to a nonempty string, the GAMS/SCIP interface opens the interactive shell of SCIP after having load the GAMS problem and parameters and passes the value of the `gams/interactive` parameter to the SCIP interactive shell.

By default, SCIP behaves as if `gams/interactive` has been set to "optimize write gamssol quit", that is, SCIP is requested to solve the problem, then to pass the solution back to GAMS, and to quit.

An example use of the SCIP interactive shell feature via GAMS is to add the following line to your SCIP options file:

```plaintext
gams/interactive = "write prob orig.lp presolve write transprob presol.mps opt write gamssol quit"
```

This instructs SCIP to write the original problem to the file `orig.lp` in LP format, to presolve the instance, to write the presolved problem to the file `presolved.mps` in MPS format, to solve the instance, to write the solution out to GAMS, and to finish.

By omitting the `quit` command, SCIPs interactive shell remains open and awaits user input. The command `help` prints a list of available commands. Note, that on Windows, GAMS need to be called with the option `interactivesolver = 1` to allow user input for the solver process.


2.2 Emphasis Settings

SCIP includes various emphasis settings, which are predefined values for a set of SCIP parameters. Such predefined settings are available for setting the effort that SCIP should spend for, e.g., presolving, separation, or heuristics.

The emphasis settings are not available as single parameters, but can be set via SCIPs interactive shell. E.g., writing `set emphasis feasibility` in the shell displays the available emphasis settings for heuristics (`aggressive`, `fast`, `off`) and expects the user to input which setting to use. Further, general emphasis settings are available in the `set emphasis` menu, some of them giving predefined settings similar to the CPLEX option `mipemphasis`.

Via the `gams/interactive` option, c.f. Section SCIP interactive shell, emphasis settings can be activated via a SCIP options file. For example, adding the option

```plaintext
gams/interactive = "set emphasis feasibility optimize write gamssol quit"
```

instructs SCIP to load the emphasis setting `feasibility` prior to optimizing the model and passing the solution back to GAMS.

The following emphasis settings are available in SCIP:

<table>
<thead>
<tr>
<th>shell command</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set emphasis easycip</code></td>
<td>use for easy problems</td>
</tr>
<tr>
<td><code>set emphasis feasibility</code></td>
<td>emphasize finding a feasible solution</td>
</tr>
<tr>
<td><code>set emphasis hardlp</code></td>
<td>use for problems with a hard LP</td>
</tr>
<tr>
<td><code>set emphasis optimality</code></td>
<td>emphasize proving optimality</td>
</tr>
<tr>
<td><code>set heuristic emphasis aggressive</code></td>
<td>use primal heuristics aggressively</td>
</tr>
<tr>
<td><code>set heuristic emphasis fast</code></td>
<td>use only fast primal heuristics</td>
</tr>
<tr>
<td><code>set heuristic emphasis off</code></td>
<td>disable all primal heuristics</td>
</tr>
</tbody>
</table>
2.3 Solution Pool

When SCIP solves a problem, it may find several solutions, whereof only the best one is available to the GAMS user via the variable level values in the GAMS model. If the option `gams/dumpsolutions` is specified, then all alternative solutions found by SCIP are written into GDX files and an index file with the name given by the this option is written.

The GAMS testlib model `dumpsol` shows an example use for this option via GUROBI. It can easily be adapted to be used with SCIP.

2.4 Solving process tracing

The option `gams/solvetrace/file` can be used to specify the name of a file where information about the progress of the branch-and-bound tree search in SCIP is stored. The file is created and updated during the solution process, so it may also be used to monitor the progress of SCIP while it still solves the model.

New entries are written periodically, depending on how many nodes have been processed or how much time has been elapsed since the last entry was written. Each entry contains information on the current primal and dual bound.

2.5 Notes on solving MINLPs with SCIP

SCIP includes capabilities to handle nonlinear functions that are specified via algebraic expressions. Therefore, external functions and not all GAMS operands are supported yet, including trigonometric ones (sin, cos, ...).

Nonconvex MINLPs are solved via a spatial branch-and-bound algorithm using linear relaxations. The tightness of this relaxation depends heavily on the variable bounds, thus tight bounds for the nonlinear variables are crucial for SCIP.

Special options for convex MINLPs

Convex MINLPs are much easier to solve for SCIP, provided it recognizes the convexity of the model. So far, only a simple convexity check is implemented in SCIP, which may not give a conclusive answer in all cases. However, setting the option `constraints/nonlinear/assumeconvex` to `TRUE` can be used to tell SCIP that it should assume all nonlinear constraints to be of convex type. This may help to improve solving times for convex MINLPs considerably.

Another useful feature especially for convex MINLPs is to enable the generation of cuts in the solution of the NLP relaxation in the root node and to consider using these cuts during the whole solution process. This is achieved by the parameters

```
constraints/quadratic/sepanlpmincont = 0
constraints/soc/sepanlpmincont = 0
constraints/nonlinear/sepanlpmincont = 0
constraints/abspower/sepanlpmincont = 0
separating/poolfreq = 1
```

3 List of SCIP Options

SCIP supports a large set of options. Sample option files can be obtained from `http://www.gams.com/~svigerske/scip3.0`. 
In the following we give a detailed list of most SCIP options.

### 3.1 gams

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gams/dumpsolutions</td>
<td>name of solutions index gdx file for writing all solutions</td>
<td></td>
</tr>
<tr>
<td>gams/indicatorfile</td>
<td>name of GAMS options file that contains definitions on indicators</td>
<td></td>
</tr>
<tr>
<td>gams/interactive</td>
<td>command to be issued to the SCIP shell instead of issuing a solve command</td>
<td></td>
</tr>
<tr>
<td>gams/mipstart</td>
<td>whether to try GAMS variable level values as initial primal solution</td>
<td>1</td>
</tr>
<tr>
<td>gams/resolvenlp</td>
<td>whether to resolve MINLP with fixed discrete variables if best solution violates some constraints</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.2 gams/solvetrace

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gams/solvetrace/file</td>
<td>name of file where to write branch-and-bound trace information too</td>
<td></td>
</tr>
<tr>
<td>gams/solvetrace/nodefreq</td>
<td>frequency in number of nodes when to write branch-and-bound trace information, 0 to disable</td>
<td>100</td>
</tr>
<tr>
<td>gams/solvetrace/timefreq</td>
<td>frequency in seconds when to write branch-and-bound trace information, 0.0 to disable</td>
<td>5</td>
</tr>
</tbody>
</table>

### 3.3 branching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/checksol</td>
<td>should LP solutions during strong branching with propagation be checked for feasibility?</td>
<td>1</td>
</tr>
<tr>
<td>branching/clamp</td>
<td>minimal relative distance of branching point to bounds when branching on a continuous variable Range: [0, 0.5]</td>
<td>0.2</td>
</tr>
<tr>
<td>branching/delaypscostupdate</td>
<td>should updating pseudo costs for continuous variables be delayed to the time after separation?</td>
<td>1</td>
</tr>
<tr>
<td>branching/firstsbschild</td>
<td>child node to be regarded first during strong branching (only with propagation): ‘u’p child, ‘d’own child, or ‘a’utomatic</td>
<td>a</td>
</tr>
<tr>
<td>branching/forceallchildren</td>
<td>should all strong branching children be regarded even if one is detected to be infeasible? (only with propagation)</td>
<td>0</td>
</tr>
<tr>
<td>branching/lpgainnormalize</td>
<td>strategy for normalization of LP gain when updating pseudocosts of continuous variables (divide by movement of ’l’p value, reduction in ’d’omain width, or reduction in domain width of ’s’ibling)</td>
<td>s</td>
</tr>
<tr>
<td>branching/preferbinary</td>
<td>should branching on binary variables be preferred?</td>
<td>0</td>
</tr>
<tr>
<td>branching/roundsbsol</td>
<td>should LP solutions during strong branching with propagation be rounded? (only when checksbsol=TRUE)</td>
<td>1</td>
</tr>
<tr>
<td>branching/scorefac</td>
<td>branching score factor to weigh downward and upward gain prediction in sum score function Range: [0, 1]</td>
<td>0.167</td>
</tr>
<tr>
<td>branching/scorefunc</td>
<td>branching score function (’s’um, ’p’rodct, ’q’uotient)</td>
<td>p</td>
</tr>
</tbody>
</table>
### 3.4 branching/allfullstrong

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/allfullstrong/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>branching/allfullstrong/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;allfullstrong&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/allfullstrong/priority</td>
<td>priority of branching rule &lt;allfullstrong&gt; Range: [-536870912, 536870911]</td>
<td>-1000</td>
</tr>
</tbody>
</table>

### 3.5 branching/cloud

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/cloud/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>branching/cloud/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;cloud&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/cloud/maxdepthunion</td>
<td>maximum depth for the union Range: [0, 65000]</td>
<td>65000</td>
</tr>
<tr>
<td>branching/cloud/maxpoints</td>
<td>maximum number of points for the cloud (-1 means no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/cloud/minsuccesrate</td>
<td>minimum success rate for the cloud Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/minsuccesunion</td>
<td>minimum success rate for the union Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/onlyF2</td>
<td>should only F2 be used?</td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/priority</td>
<td>priority of branching rule &lt;cloud&gt; Range: [-536870912, 536870911]</td>
<td>0</td>
</tr>
<tr>
<td>branching/cloud/usecloud</td>
<td>should a cloud of points be used?</td>
<td>1</td>
</tr>
<tr>
<td>branching/cloud/useunion</td>
<td>should the union of candidates be used?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.6 branching/fullstrong

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/fullstrong/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>branching/fullstrong/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;fullstrong&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/fullstrong/maxproprounds</td>
<td>maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings)</td>
<td>-2</td>
</tr>
<tr>
<td>branching/fullstrong/priority</td>
<td>priority of branching rule &lt;fullstrong&gt; Range: [-536870912, 536870911]</td>
<td>0</td>
</tr>
</tbody>
</table>
3.7 branching/inference

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/inference/conflictweight</td>
<td>weight in score calculations for conflict score</td>
<td>1000</td>
</tr>
<tr>
<td>branching/inference/cutoffweight</td>
<td>weight in score calculations for cutoff score</td>
<td>1</td>
</tr>
<tr>
<td>branching/inference/fractionals</td>
<td>should branching on LP solution be restricted to the fractional variables?</td>
<td>1</td>
</tr>
<tr>
<td>branching/inference/inferenceweight</td>
<td>weight in score calculations for inference score</td>
<td>1</td>
</tr>
<tr>
<td>branching/inference/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/inference/maxdepth</td>
<td>maximal depth level, up to which branching rule $&lt;$inference$&gt;$ should be used (-1 for no limit)</td>
<td>$\leq 1$</td>
</tr>
<tr>
<td>branching/inference/priority</td>
<td>priority of branching rule $&lt;$inference$&gt;$</td>
<td>1000</td>
</tr>
<tr>
<td>branching/inference/reliablescore</td>
<td>weight in score calculations for conflict score</td>
<td>0.001</td>
</tr>
<tr>
<td>branching/inference/useweightedsum</td>
<td>should a weighted sum of inference, conflict and cutoff weights be used?</td>
<td>1</td>
</tr>
</tbody>
</table>

3.8 branching/leastinf

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/leastinf/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/leastinf/maxdepth</td>
<td>maximal depth level, up to which branching rule $&lt;$leastinf$&gt;$ should be used (-1 for no limit)</td>
<td>$\leq 1$</td>
</tr>
<tr>
<td>branching/leastinf/priority</td>
<td>priority of branching rule $&lt;$leastinf$&gt;$</td>
<td>50</td>
</tr>
</tbody>
</table>

3.9 branching/mostinf

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/mostinf/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/mostinf/maxdepth</td>
<td>maximal depth level, up to which branching rule $&lt;$mostinf$&gt;$ should be used (-1 for no limit)</td>
<td>$\leq 1$</td>
</tr>
</tbody>
</table>
### 3.10 branching/pscost

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/pscost/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/pscost/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;pscost&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/pscost/maxscoreweight</td>
<td>weight for maximum of scores of a branching candidate when building weighted sum of min/max/sum of scores</td>
<td>1.3</td>
</tr>
<tr>
<td>branching/pscost/minscoreweight</td>
<td>weight for minimum of scores of a branching candidate when building weighted sum of min/max/sum of scores</td>
<td>0.8</td>
</tr>
<tr>
<td>branching/pscost/narymaxdepth</td>
<td>maximal depth where to do n-ary branching, -1 to turn off</td>
<td>-1</td>
</tr>
<tr>
<td>branching/pscost/naryminwidth</td>
<td>minimal domain width in children when doing n-ary branching, relative to global bounds</td>
<td>0.001</td>
</tr>
<tr>
<td>branching/pscost/narywidthfactor</td>
<td>factor of domain width in n-ary branching when creating nodes with increasing distance from branching value</td>
<td>2</td>
</tr>
<tr>
<td>branching/pscost/nchildren</td>
<td>number of children to create in n-ary branching</td>
<td>2</td>
</tr>
<tr>
<td>branching/pscost/priority</td>
<td>priority of branching rule &lt;pscost&gt;</td>
<td>2000</td>
</tr>
<tr>
<td>branching/pscost/strategy</td>
<td>strategy for utilizing pseudo-costs of external branching candidates (multiply as in pseudo costs ‘u’pdate rule, or by ’d’omain reduction, or by domain reduction of ’s’ibling, or by ’v’ariable score)</td>
<td>u</td>
</tr>
<tr>
<td>branching/pscost/sumscoreweight</td>
<td>weight for sum of scores of a branching candidate when building weighted sum of min/max/sum of scores</td>
<td>0.1</td>
</tr>
</tbody>
</table>

### 3.11 branching/random

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>branching/random/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>branching/random/maxdepth</td>
<td>maximal depth level, up to which branching rule &lt;random&gt; should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>branching/random/priority</td>
<td>priority of branching rule &lt;random&gt;</td>
<td>-100000</td>
</tr>
<tr>
<td>branching/random/seed</td>
<td>initial random seed value</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.12 branching/relpscost

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>branching/relpscost/conflictlengthweight</code></td>
<td>weight in score calculations for conflict length score</td>
<td>0</td>
</tr>
<tr>
<td><code>branching/relpscost/conflictweight</code></td>
<td>weight in score calculations for conflict score</td>
<td>0.01</td>
</tr>
<tr>
<td><code>branching/relpscost/cutoffweight</code></td>
<td>weight in score calculations for cutoff score</td>
<td>0.0001</td>
</tr>
<tr>
<td><code>branching/relpscost/inferenceweight</code></td>
<td>weight in score calculations for inference score</td>
<td>0.0001</td>
</tr>
<tr>
<td><code>branching/relpscost/initcand</code></td>
<td>maximal number of candidates initialized with strong branching per node</td>
<td>100</td>
</tr>
<tr>
<td><code>branching/relpscost/inititer</code></td>
<td>iteration limit for strong branching initializations of pseudo cost entries (0: auto)</td>
<td>0</td>
</tr>
<tr>
<td><code>branching/relpscost/maxbdchgs</code></td>
<td>maximal number of bound tightenings before the node is reevaluated (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td><code>branching/relpscost/maxbounddist</code></td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td><code>branching/relpscost/maxdepth</code></td>
<td>maximal depth level, up to which branching rule <code>&lt;relpscost&gt;</code> should be used (-1 for no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><code>branching/relpscost/maxlookahead</code></td>
<td>maximal number of further variables evaluated without better score</td>
<td>9</td>
</tr>
<tr>
<td><code>branching/relpscost/maxproprounds</code></td>
<td>maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings)</td>
<td>-2</td>
</tr>
<tr>
<td><code>branching/relpscost/maxreliable</code></td>
<td>maximal value for minimum pseudo cost size to regard pseudo cost value as reliable</td>
<td>5</td>
</tr>
<tr>
<td><code>branching/relpscost/minreliable</code></td>
<td>minimal value for minimum pseudo cost size to regard pseudo cost value as reliable</td>
<td>1</td>
</tr>
<tr>
<td><code>branching/relpscost/priority</code></td>
<td>priority of branching rule <code>&lt;relpscost&gt;</code> Range: [-536870912, 536870911]</td>
<td>10000</td>
</tr>
<tr>
<td><code>branching/relpscost/probingbounds</code></td>
<td>should valid bounds be identified in a probing-like fashion during strong branching (only with propagation)?</td>
<td>1</td>
</tr>
<tr>
<td><code>branching/relpscost/pscostweight</code></td>
<td>weight in score calculations for pseudo cost score</td>
<td>1</td>
</tr>
<tr>
<td><code>branching/relpscost/sbiterofs</code></td>
<td>additional number of allowed strong branching LP iterations</td>
<td>100000</td>
</tr>
<tr>
<td><code>branching/relpscost/sbiterquot</code></td>
<td>maximal fraction of strong branching LP iterations compared to node relaxation LP iterations</td>
<td>0.5</td>
</tr>
</tbody>
</table>

### 3.13 conflict

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>conflict/allowlocal</code></td>
<td>should conflict constraints be generated that are only valid locally?</td>
<td>1</td>
</tr>
<tr>
<td><code>conflict/conflictgraphweight</code></td>
<td>the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict graph Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td><code>conflict/conflictweight</code></td>
<td>the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td><code>conflict/deepthscorefac</code></td>
<td>score factor for depth level in bound relaxation heuristic of LP analysis</td>
<td>1</td>
</tr>
<tr>
<td><code>conflict/dynamic</code></td>
<td>should the conflict constraints be subject to aging?</td>
<td>1</td>
</tr>
<tr>
<td><code>conflict/enable</code></td>
<td>should conflict analysis be enabled?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>conflict/fuiplevels</td>
<td>number of depth levels up to which first UIP’s are used in conflict analysis (-1: use All-FirstUIP rule)</td>
<td>-1</td>
</tr>
<tr>
<td>conflict/fullshortenconflict</td>
<td>try to shorten the whole conflict set or terminate early (depending on the 'maxvarsdetectimpliedbounds' parameter)</td>
<td>1</td>
</tr>
<tr>
<td>conflict/ignorerelaxedbd</td>
<td>should relaxed bounds be ignored?</td>
<td>0</td>
</tr>
<tr>
<td>conflict/interconss</td>
<td>maximal number of intermediate conflict constraints generated in conflict graph (-1: use every intermediate constraint)</td>
<td>-1</td>
</tr>
<tr>
<td>conflict/keepreprop</td>
<td>should constraints be kept for repropagation even if they are too long?</td>
<td>1</td>
</tr>
<tr>
<td>conflict/lpiterations</td>
<td>maximal number of LP iterations in each LP resolving loop (-1: no limit)</td>
<td>10</td>
</tr>
<tr>
<td>conflict/maxconss</td>
<td>maximal number of conflict constraints accepted at an infeasible node (-1: use all generated conflict constraints)</td>
<td>10</td>
</tr>
<tr>
<td>conflict/maxlploops</td>
<td>maximal number of LP resolving loops during conflict analysis (-1: no limit)</td>
<td>2</td>
</tr>
<tr>
<td>conflict/maxvarsdetectimpliedbounds</td>
<td>maximal number of variables to try to detect global bound implications and shorten the whole conflict set (0: disabled)</td>
<td>250</td>
</tr>
<tr>
<td>conflict/maxvarsfac</td>
<td>maximal fraction of variables involved in a conflict constraint</td>
<td>0.1</td>
</tr>
<tr>
<td>conflict/minmaxvars</td>
<td>minimal absolute maximum of variables involved in a conflict constraint</td>
<td>30</td>
</tr>
<tr>
<td>conflict/preferbinary</td>
<td>should binary conflicts be preferred?</td>
<td>0</td>
</tr>
<tr>
<td>conflict/reconvlevels</td>
<td>number of depth levels up to which UIP reconvergence constraints are generated (-1: generate reconvergence constraints in all depth levels)</td>
<td>-1</td>
</tr>
<tr>
<td>conflict/removable</td>
<td>should the conflict’s relaxations be subject to LP aging and cleanup?</td>
<td>1</td>
</tr>
<tr>
<td>conflict/repropagate</td>
<td>should earlier nodes be repropagated in order to replace branching decisions by deductions?</td>
<td>1</td>
</tr>
<tr>
<td>conflict/restartfac</td>
<td>factor to increase restartnum with after each restart</td>
<td>1.5</td>
</tr>
<tr>
<td>conflict/restartnum</td>
<td>number of successful conflict analysis calls that trigger a restart (0: disable conflict restarts)</td>
<td>0</td>
</tr>
<tr>
<td>conflict/scorefac</td>
<td>factor to decrease importance of variables’ earlier conflict scores</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-06, 1]</td>
<td></td>
</tr>
<tr>
<td>conflict/separate</td>
<td>should the conflict constraints be separated?</td>
<td>1</td>
</tr>
<tr>
<td>conflict/settlelocal</td>
<td>should conflict constraints be attached only to the local subtree where they can be useful?</td>
<td>0</td>
</tr>
<tr>
<td>conflict/useboundlp</td>
<td>should bound exceeding LP conflict analysis be used?</td>
<td>0</td>
</tr>
<tr>
<td>conflict/useinflp</td>
<td>should infeasible LP conflict analysis be used?</td>
<td>1</td>
</tr>
<tr>
<td>conflict/useprop</td>
<td>should propagation conflict analysis be used?</td>
<td>1</td>
</tr>
<tr>
<td>conflict/usepseudo</td>
<td>should pseudo solution conflict analysis be used?</td>
<td>1</td>
</tr>
<tr>
<td>conflict/usesb</td>
<td>should infeasible/bound exceeding strong branching conflict analysis be used?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.14 conflict/bounddisjunction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict/bounddisjunction/continuousfrac</td>
<td>maximal percentage of continuous variables within a conflict Range: [0, 1]</td>
<td>0.4</td>
</tr>
<tr>
<td>conflict/bounddisjunction/priority</td>
<td>priority of conflict handler &lt;bounddisjunction&gt;</td>
<td>-3000000</td>
</tr>
</tbody>
</table>
3.15  conflict/indicatorconflict

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict/indicatorconflict/priority</td>
<td>priority of conflict handler &lt;indicatorconflict&gt;</td>
<td>200000</td>
</tr>
</tbody>
</table>

3.16  conflict/linear

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict/linear/priority</td>
<td>priority of conflict handler &lt;linear&gt;</td>
<td>-1000000</td>
</tr>
</tbody>
</table>

3.17  conflict/logicor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict/logicor/priority</td>
<td>priority of conflict handler &lt;logicor&gt;</td>
<td>800000</td>
</tr>
</tbody>
</table>

3.18  conflict/setppc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict/setppc/priority</td>
<td>priority of conflict handler &lt;setppc&gt;</td>
<td>700000</td>
</tr>
</tbody>
</table>

3.19  constraints

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/agelimit</td>
<td>maximum age an unnecessary constraint can reach before it is deleted (0: dynamic, -1: keep all constraints)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/disableenfops</td>
<td>should enforcement of pseudo solution be disabled?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/obsoleteage</td>
<td>age of a constraint after which it is marked obsolete (0: dynamic, -1 do not mark constraints obsolete)</td>
<td>-1</td>
</tr>
</tbody>
</table>

3.20  constraints/SOS1

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/SOS1/branchnonzeros</td>
<td>Branch on SOS constraint with most number of nonzeros?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/branchsos</td>
<td>Use SOS1 branching in enforcing (otherwise leave decision to branching rules)?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/branchweight</td>
<td>Branch on SOS cons. with highest nonzero-variable weight for branching (needs branchnonzeros = false)?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/SOS1/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/SOS1/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS1/seapafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS1/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.21 constraints/SOS2

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/SOS2/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS2/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS2/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS2/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/SOS2/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/SOS2/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/SOS2/seapafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/SOS2/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.22 constraints/abspower

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/abspower/addvarboundcons</td>
<td>should variable bound constraints be added for derived variable bounds?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/abspower/branchminconverror</td>
<td>whether to compute branching point such that the convexification error is minimized (after branching on 0.0)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/abspower/cutmaxrange</td>
<td>maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation</td>
<td>1e+07</td>
</tr>
<tr>
<td>constraints/abspower/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/abspower/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/abspower/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/abspower/dualpresolve</td>
<td>should dual presolve be applied?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/abspower/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/abspower/enfocutsremovable</td>
<td>are cuts added during enforcement removable from the LP in the same node?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/abspower/linfeasshift</td>
<td>whether to try to make solutions in check function feasible by shifting the linear variable z</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/abspower/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/abspower/minefficacyenfofac</td>
<td>minimal target efficacy of a cut in order to add it to relaxation during enforcement as factor of feasibility tolerance (may be ignored)</td>
<td>2</td>
</tr>
<tr>
<td>constraints/abspower/minefficacysepa</td>
<td>minimal efficacy for a cut to be added to the LP during separation; overwrites separating/efficacy</td>
<td>0.0001</td>
</tr>
<tr>
<td>constraints/abspower/preferzerobranch</td>
<td>how much to prefer branching on 0.0 when sign of variable is not fixed yet: 0 no preference, 1 prefer if LP solution will be cutoff in both child nodes, 2 prefer always, 3 ensure always</td>
<td>1</td>
</tr>
<tr>
<td>constraints/abspower/projectrefpoint</td>
<td>whether to project the reference point when linearizing an absolute power constraint in a convex region</td>
<td>1</td>
</tr>
<tr>
<td>constraints/abspower/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/abspower/scaling</td>
<td>whether scaling of infeasibility is ’o’ff, by sup-norm of function ’g’radient, or by left/right hand ’s’ide</td>
<td>o</td>
</tr>
<tr>
<td>constraints/abspower/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/abspower/sepenboundsonly</td>
<td>whether to separate linearization cuts only in the variable bounds (does not affect enforcement)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/abspower/sepanlpmincont</td>
<td>minimal required fraction of continuous variables in problem to use solution of NLP relaxation in root for separation</td>
<td>1</td>
</tr>
<tr>
<td>constraints/abspower/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>15</td>
</tr>
</tbody>
</table>

### 3.23 constraints/and

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/and/aggrlinearization</td>
<td>should an aggregated linearization be used?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/dualpresolving</td>
<td>should dual presolving be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/and/enforcecuts</td>
<td>should cuts be separated during LP enforcing?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/linearize</td>
<td>should the AND-constraint get linearized and removed (in presolving)?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/and/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/and/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/and/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
</tbody>
</table>
### 3.24 constraints/bivariate

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/bivariate/cutmaxrange</td>
<td>maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation</td>
<td>1e+07</td>
</tr>
<tr>
<td>constraints/bivariate/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bivariate/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bivariate/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bivariate/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/bivariate/enfocutsremovable</td>
<td>are cuts added during enforcement removable from the LP in the same node?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bivariate/linfeasshift</td>
<td>whether to try to make solutions in check function feasible by shifting a linear variable (esp. useful if constraint was actually objective function)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/bivariate/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/bivariate/maxproprounds</td>
<td>limit on number of propagation rounds for a single constraint within one round of SCIP propagation</td>
<td>1</td>
</tr>
<tr>
<td>constraints/bivariate/minefficacyenfo</td>
<td>minimal target efficacy of a cut in order to add it to relaxation during enforcement (may be ignored)</td>
<td>2e-06</td>
</tr>
<tr>
<td>constraints/bivariate/minefficacysepa</td>
<td>minimal efficacy for a cut to be added to the LP during separation; overwrites separating/efficacy</td>
<td>0.0001</td>
</tr>
<tr>
<td>constraints/bivariate/ninitprefsepa</td>
<td>number of reference points in each direction where to compute linear support for envelope in LP initialization</td>
<td>3</td>
</tr>
<tr>
<td>constraints/bivariate/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/bivariate/scaling</td>
<td>whether scaling of infeasibility is 'o'ff, by sup-norm of function 'g'radient, or by left/right hand 's'ide</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bivariate/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/bivariate/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.25 constraints/bounddisjunction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/bounddisjunction/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
</tbody>
</table>
### constraints/bounddisjunction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/bounddisjunction/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bounddisjunction/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/bounddisjunction/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/bounddisjunction/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>−1</td>
</tr>
<tr>
<td>constraints/bounddisjunction/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/bounddisjunction/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>−1</td>
</tr>
<tr>
<td>constraints/bounddisjunction/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>

### constraints/indicator

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/indicator/addcoupling</td>
<td>Add coupling constraints if big-M is small enough?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/addcouplingcons</td>
<td>Add initial coupling inequalities as linear constraints, if ‘addcoupling’ is true?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/addopposite</td>
<td>Add opposite inequality in nodes in which the binary variable has been fixed to 0?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/branchindicators</td>
<td>Branch on indicator constraints in enforcing?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/conflictsupgrade</td>
<td>Try to upgrade bounddisjunction conflicts by replacing slack variables?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/dualreductions</td>
<td>Should dual reduction steps be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/indicator/enforcecuts</td>
<td>In enforcing try to generate cuts (only if sepaalternativelp is true)?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/forcererestart</td>
<td>Force restart if we have a max FS instance and gap is 1?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/generatebilinear</td>
<td>Do not generate indicator constraint, but a bilinear constraint instead?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/genlogicor</td>
<td>Generate logicor constraints instead of cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/maxconditionalltp</td>
<td>maximum estimated condition of the solution basis matrix of the alternative LP to be trustworthy (0.0 to disable check)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/maxcouplingvalue</td>
<td>maximum coefficient for binary variable in coupling constraint Range: [0, 1e+09]</td>
<td>10000</td>
</tr>
<tr>
<td>constraints/indicator/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>−1</td>
</tr>
<tr>
<td>constraints/indicator/maxsepacuts</td>
<td>maximal number of cuts separated per separation round</td>
<td>100</td>
</tr>
</tbody>
</table>
###constraints/indicator/

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/indicator/maxsepacutsroot</td>
<td>maximal number of cuts separated per separation round in the root node</td>
<td>2000</td>
</tr>
<tr>
<td>constraints/indicator/holincconscont</td>
<td>Decompose problem (do not generate linear constraint if all variables are continuous)?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/removeindicat...</td>
<td>Remove indicator constraint if corresponding variable bound constraint has been added?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/restartfrac</td>
<td>fraction of binary variables that need to be fixed before restart occurs (in forcerestart) Range: [0, 1]</td>
<td>0.9</td>
</tr>
<tr>
<td>constraints/indicator/scaleslackvar</td>
<td>Scale slack variable coefficient at construction time?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/sepaalternativelp</td>
<td>Separate using the alternative LP?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/sepacouplingc...</td>
<td>Should the coupling inequalities be separated dynamically?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/sepacouplinglocal</td>
<td>Allow to use local bounds in order to separated coupling inequalities?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/sepacouplingvalue</td>
<td>maximum coefficient for binary variable in separated coupling constraint Range: [0, 1e+09]</td>
<td>10000</td>
</tr>
<tr>
<td>constraints/indicator/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>10</td>
</tr>
<tr>
<td>constraints/indicator/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/trysolfromcover</td>
<td>Try to construct a feasible solution from a cover?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/trysolutions</td>
<td>Try to make solutions feasible by setting indicator variables?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/indicator/updatebounds</td>
<td>Update bounds of original variables for separation?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/indicator/useotherconss</td>
<td>Collect other constraints to alternative LP?</td>
<td>0</td>
</tr>
</tbody>
</table>

###3.27 constraints/integral

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/integral/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/integral/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/integral/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/integral/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/integral/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/integral/propfreq</td>
<td>Frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/integral/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/integral/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>
### 3.28 constraints/knapsack

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/knapsack/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/detectcutoffbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/detectlowerbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/disaggregation</td>
<td>should disaggregation of knapsack constraints be allowed in preprocessing?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/dualpresolving</td>
<td>should dual presolving steps be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/knapsack/maxcardbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for separating knapsack cuts Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/knapsack/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>constraints/knapsack/maxroundsroot</td>
<td>maximal number of separation rounds per node in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/knapsack/maxsepacuts</td>
<td>maximal number of cuts separated per separation round</td>
<td>50</td>
</tr>
<tr>
<td>constraints/knapsack/maxsepacutsroot</td>
<td>maximal number of cuts separated per separation round in the root node</td>
<td>200</td>
</tr>
<tr>
<td>constraints/knapsack/negatedclique</td>
<td>should negated clique information be used in solving process</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/sepacardfreq</td>
<td>multiplier on separation frequency, how often knapsack cuts are separated (-1: never, 0: only at root)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/knapsack/simplifyinequalities</td>
<td>should presolving try to simplify knapsacks</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
<tr>
<td>constraints/knapsack/usegubs</td>
<td>should GUB information be used for separation?</td>
<td>0</td>
</tr>
</tbody>
</table>
### 3.29  constraints/linear

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/linear/aggregatevariables</td>
<td>should presolving search for aggregations in equations</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/checkrelmaxabs</td>
<td>should the violation for a constraint with side 0.0 be checked relative to 1.0 (FALSE) or to the maximum absolute value in the activity (TRUE)?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/detectcutoffbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/detectlowerbound</td>
<td>should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/detectpartialobjective</td>
<td>should presolving try to detect subsets of constraints parallel to the objective function?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/dualpresolving</td>
<td>should dual presolving steps be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/linear/maxaggrnomscale</td>
<td>maximal allowed relative gain in maximum norm for constraint aggregation (0.0: disable constraint aggregation)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/maxcardbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for separating knapsack cardinality cuts Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/linear/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>constraints/linear/maxroundsroot</td>
<td>maximal number of separation rounds per node in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/linear/maxsepacuts</td>
<td>maximal number of cuts separated per separation round</td>
<td>50</td>
</tr>
<tr>
<td>constraints/linear/maxsepacutsroot</td>
<td>maximal number of cuts separated per separation round in the root node</td>
<td>200</td>
</tr>
<tr>
<td>constraints/linear/mingainpermmincomparisons</td>
<td>minimal gain per minimal pairwise presolve comparisons to repeat pairwise comparison round Range: [0, 1]</td>
<td>1e-06</td>
</tr>
<tr>
<td>constraints/linear/nmincomparisons</td>
<td>number for minimal pairwise presolve comparisons</td>
<td>200000</td>
</tr>
<tr>
<td>constraints/linear/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/linear/sepfreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/separateall</td>
<td>should all constraints be subject to cardinality cut generation instead of only the ones with non-zero dual value?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/linear/simplifyinequalities</td>
<td>should presolving try to simplify inequalities</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/sortvars</td>
<td>apply binaries sorting in decre. order of coeff abs value?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/tightenboundsfreq</td>
<td>multiplier on propagation frequency, how often the bounds are tightened (-1: never, 0: only at root)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.30 constraints/linear/upgrade

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/linear/upgrade/knapsack</td>
<td>enable linear upgrading for constraint handler &lt;knapsack&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/upgrade/logicor</td>
<td>enable linear upgrading for constraint handler &lt;logicor&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/upgrade/setppc</td>
<td>enable linear upgrading for constraint handler &lt;setppc&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/linear/upgrade/varbound</td>
<td>enable linear upgrading for constraint handler &lt;varbound&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.31 constraints/logicor

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/logicor/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/logicor/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/logicor/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/logicor/dualpresolving</td>
<td>should dual presolving steps be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/logicor/implications</td>
<td>should implications/cliques be used in presolving</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/logicor/negatedclique</td>
<td>should negated clique information be used in presolving</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/presolusenem inbound</td>
<td>should hash table be used for detecting redundant constraints in advance</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/sepfreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/logicor/strengthen</td>
<td>should pairwise constraint comparison try to strengthen constraints by removing superfluous non-zeros?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/logicor/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>
### 3.32 constraints/nonlinear

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/nonlinear/assumeconvex</td>
<td>whether to assume that nonlinear functions in inequalities ((\leq)) are convex (disables reformulation)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/cutmaxrange</td>
<td>maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation</td>
<td>1e+07</td>
</tr>
<tr>
<td>constraints/nonlinear/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/nonlinear/enfocutsremovable</td>
<td>are cuts added during enforcement removable from the LP in the same node?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/linfeasshift</td>
<td>whether to try to make solutions in check function feasible by shifting a linear variable (esp. useful if constraint was actually objective function)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/maxexpansionexponent</td>
<td>maximal exponent where still expanding non-monomial polynomials in expression simplification</td>
<td>2</td>
</tr>
<tr>
<td>constraints/nonlinear/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/nonlinear/maxproprounds</td>
<td>limit on number of propagation rounds for a single constraint within one round of SCIP propagation</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/minefficacyenfofac</td>
<td>minimal target efficacy of a cut in order to add it to relaxation during enforcement as a factor of the feasibility tolerance (may be ignored)</td>
<td>2</td>
</tr>
<tr>
<td>constraints/nonlinear/minefficacysepa</td>
<td>minimal efficacy for a cut to be added to the LP during separation; overwrites separating/efficacy</td>
<td>0.0001</td>
</tr>
<tr>
<td>constraints/nonlinear/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/reformulate</td>
<td>whether to reformulate expression graph</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/scaling</td>
<td>whether scaling of infeasibility is ‘o’ff, by sup-norm of function ’g’radient, or by left/right hand ’s’ide</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/sepaefreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/sepanlpmincont</td>
<td>minimal required fraction of continuous variables in problem to use solution of NLP relaxation in root for separation Range: [0, 2]</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.33 constraints/nonlinear/upgrade
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/nonlinear/upgrade/abspower</td>
<td>enable nonlinear upgrading for constraint handler &lt;abspower&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/upgrade/and</td>
<td>enable nonlinear upgrading for constraint handler &lt;and&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/upgrade/bivariate</td>
<td>enable nonlinear upgrading for constraint handler &lt;bivariate&gt;</td>
<td>0</td>
</tr>
<tr>
<td>constraints/nonlinear/upgrade/linear</td>
<td>enable nonlinear upgrading for constraint handler &lt;linear&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/nonlinear/upgrade/quadratic</td>
<td>enable nonlinear upgrading for constraint handler &lt;quadratic&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.34 constraints/quadratic

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/quadratic/binreforminitial</td>
<td>whether to make non-varbound linear constraints added due to replacing products with binary variables</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/binreformmaxcoef</td>
<td>limit (as factor on 1/feastol) on coefficients and coeff. range in linear constraints created when replacing products with binary variables</td>
<td>0.0001</td>
</tr>
<tr>
<td>constraints/quadratic/checkcurvature</td>
<td>whether multivariate quadratic functions should be checked for convexity/concavity</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/checkfactorable</td>
<td>whether constraint functions should be checked to be factorable</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/cutmaxrange</td>
<td>maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation</td>
<td>1e+07</td>
</tr>
<tr>
<td>constraints/quadratic/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/disaggregate</td>
<td>whether to disaggregate quadratic parts that decompose into a sum of non-overlapping quadratic terms</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/quadratic/empathy4and</td>
<td>empathy level for using the AND constraint handler: 0 always avoid using AND; 1 use AND sometimes; 2 use AND as often as possible Range: [0, 2]</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/enfocutsremovable</td>
<td>are cuts added during enforcement removable from the LP in the same node?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/enfolplimit</td>
<td>maximum number of enforcement rounds before declaring the LP relaxation infeasible (-1: no limit); WARNING: changing this parameter might lead to incorrect results!</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/quadratic/linearizeheursol</td>
<td>whether linearizations of convex quadratic constraints should be added to cutpool in a solution found by some heuristic</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/linfeasshift</td>
<td>whether to try to make solutions in check function feasible by shifting a linear variable (esp. useful if constraint was actually objective function)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/quadratic/maxproprounds</td>
<td>limit on number of propagation rounds for a single constraint within one round of SCIP propagation during solve</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>constraints/quadratic/maxproproundspresolve</td>
<td>limit on number of propagation rounds for a single constraint within one round of SCIP presolve</td>
<td>10</td>
</tr>
<tr>
<td>constraints/quadratic/minefficacyenfoc</td>
<td>minimal target efficacy of a cut in order to add it to relaxation during enforcement as a factor of the feasibility tolerance (may be ignored)</td>
<td>2</td>
</tr>
<tr>
<td>constraints/quadratic/minefficacysepa</td>
<td>minimal efficacy for a cut to be added to the LP during separation; overwrites separating/efficacy</td>
<td>0.0001</td>
</tr>
<tr>
<td>constraints/quadratic/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/replacebinaryprod</td>
<td>max. length of linear term which when multiplied with a binary variables is replaced by an auxiliary variable and a linear reformulation (0 to turn off)</td>
<td>maxint</td>
</tr>
<tr>
<td>constraints/quadratic/scaling</td>
<td>whether scaling of infeasibility is 'off, by sup-norm of function 'gradient, or by left/right hand 'side</td>
<td>o</td>
</tr>
<tr>
<td>constraints/quadratic/sepaefreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/sepanlpmincont</td>
<td>minimal required fraction of continuous variables in problem to use solution of NLP relaxation in root for separation</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.35 constraints/quadratic/upgrade

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/quadratic/upgrade/abspower</td>
<td>enable quadratic upgrading for constraint handler &lt;abspower&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/upgrade/bivariate</td>
<td>enable quadratic upgrading for constraint handler &lt;bivariate&gt;</td>
<td>0</td>
</tr>
<tr>
<td>constraints/quadratic/upgrade/bounddisjunction</td>
<td>enable quadratic upgrading for constraint handler &lt;bounddisjunction&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/upgrade/linear</td>
<td>enable quadratic upgrading for constraint handler &lt;linear&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/upgrade/setppc</td>
<td>enable quadratic upgrading for constraint handler &lt;setppc&gt;</td>
<td>1</td>
</tr>
<tr>
<td>constraints/quadratic/upgrade/soc</td>
<td>enable quadratic upgrading for constraint handler &lt;soc&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.36 constraints/setppc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/setppc/addvariablesascliques</td>
<td>should we try to generate extra cliques out of all binary variables to maybe fasten redundant constraint detection</td>
<td>0</td>
</tr>
<tr>
<td>constraints/setppc/cliquelifting</td>
<td>should we try to lift variables into other clique constraints, fix variables, aggregate them, and also shrink the amount of variables in clique constraints</td>
<td>0</td>
</tr>
<tr>
<td>constraints/setppc/cliqueshrinking</td>
<td>should we try to shrink the number of variables in a clique constraints, by replacing more than one variable by only one</td>
<td>1</td>
</tr>
</tbody>
</table>
### 3.37 constraints/soc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/soc/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/soc/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/soc/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/soc/dualpresolving</td>
<td>should dual presolving steps be performed?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/soc/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/soc/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/soc/pseudobranches</td>
<td>number of children created in pseudo branching (0: disable pseudo branching)</td>
<td>2</td>
</tr>
<tr>
<td>constraints/soc/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/soc/presolusehashing</td>
<td>should hash table be used for detecting redundant constraints in advance</td>
<td>1</td>
</tr>
<tr>
<td>constraints/soc/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/soc/sepafrq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/soc/timingmask</td>
<td>timing when constraint propagation should be called</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [1, 15]</td>
<td></td>
</tr>
<tr>
<td>constraints/soc/enfocutsremovable</td>
<td>are cuts added during enforcement removable from the LP in the same node?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/soc/glineur</td>
<td>whether the Glineur Outer Approximation should be used instead of Ben-Tal Nemirovski</td>
<td>1</td>
</tr>
<tr>
<td>constraints/soc/linfeasshift</td>
<td>whether to try to make solutions feasible in check by shifting the variable on the right hand side</td>
<td>1</td>
</tr>
<tr>
<td>constraints/soc/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/soc/minefficacy</td>
<td>minimal efficacy of a cut to be added to LP in separation</td>
<td>0.0001</td>
</tr>
<tr>
<td>constraints/soc/nauxvars</td>
<td>number of auxiliary variables to use when creating a linear outer approx. of a SOC3 constraint; 0 to turn off</td>
<td>0</td>
</tr>
</tbody>
</table>
### 3.38 constraints/varbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraints/varbound/delaypresol</td>
<td>should presolving method be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/varbound/delayprop</td>
<td>should propagation method be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/varbound/delaysepa</td>
<td>should separation method be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>constraints/varbound/eagerfreq</td>
<td>frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation)</td>
<td>100</td>
</tr>
<tr>
<td>constraints/varbound/maxlpcoef</td>
<td>maximum coefficient in varbound constraint to be added as a row into LP</td>
<td>1e+09</td>
</tr>
<tr>
<td>constraints/varbound/maxprerounds</td>
<td>maximal number of presolving rounds the constraint handler participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>constraints/varbound/presolpairwise</td>
<td>should pairwise constraint comparison be performed in presolving?</td>
<td>1</td>
</tr>
<tr>
<td>constraints/varbound/propfreq</td>
<td>frequency for propagating domains (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/varbound/sepafreq</td>
<td>frequency for separating cuts (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>constraints/varbound/timingmask</td>
<td>timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>1</td>
</tr>
<tr>
<td>constraints/varbound/usebdwidening</td>
<td>should bound widening be used in conflict analysis?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.39 display

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/freq</td>
<td>frequency for displaying node information lines</td>
<td>100</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>display/headerfreq</td>
<td>frequency for displaying header lines (every n'th node information line)</td>
<td>15</td>
</tr>
<tr>
<td>display/lpinfo</td>
<td>should the LP solver display status messages?</td>
<td>0</td>
</tr>
<tr>
<td>display/statistics</td>
<td>whether to print statistics on a solve</td>
<td>0</td>
</tr>
<tr>
<td>display/verblevel</td>
<td>verbosity level of output</td>
<td>4</td>
</tr>
<tr>
<td>display/width</td>
<td>maximal number of characters in a node information line</td>
<td>139 (80 for Windows without IDE)</td>
</tr>
</tbody>
</table>

### 3.40 display/avgdualbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/avgdualbound/active</td>
<td>display activation status of display column &lt;avgdualbound&gt; (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.41 display/conflicts

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/conflicts/active</td>
<td>display activation status of display column &lt;conflicts&gt; (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.42 display/conss

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/conss/active</td>
<td>display activation status of display column &lt;conss&gt; (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.43 display/curcols

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/curcols/active</td>
<td>display activation status of display column &lt;curcols&gt; (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.44 display/curconss

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/curconss/active</td>
<td>display activation status of display column &lt;curconss&gt; (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.45 display/curdualbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
### 3.46 display/currows

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/currows/active</td>
<td>display activation status of display column <code>&lt;currows&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
</tbody>
</table>

Range: [0, 2]

### 3.47 display/cutoffbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/cutoffbound/active</td>
<td>display activation status of display column <code>&lt;cutoffbound&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
</tbody>
</table>

Range: [0, 2]

### 3.48 display/cuts

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/cuts/active</td>
<td>display activation status of display column <code>&lt;cuts&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
</tbody>
</table>

Range: [0, 2]

### 3.49 display/depth

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/depth/active</td>
<td>display activation status of display column <code>&lt;depth&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
</tbody>
</table>

Range: [0, 2]

### 3.50 display/dualbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/dualbound/active</td>
<td>display activation status of display column <code>&lt;dualbound&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
</tbody>
</table>

Range: [0, 2]

### 3.51 display/estimate

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/estimate/active</td>
<td>display activation status of display column <code>&lt;estimate&gt;</code> (0: off, 1: auto, 2: on)</td>
<td>1</td>
</tr>
</tbody>
</table>

Range: [0, 2]

### 3.52 display/feasST
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/feasST/active</td>
<td>display activation status of display column <code>&lt;feasST&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.53 display/gap

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/gap/active</td>
<td>display activation status of display column <code>&lt;gap&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.54 display/lpavgiterations

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpavgiterations/active</td>
<td>display activation status of display column <code>&lt;lpavgiterations&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1 (0 for Windows without IDE)</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.55 display/lpcond

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpcond/active</td>
<td>display activation status of display column <code>&lt;lpcond&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.56 display/lpiterations

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpiterations/active</td>
<td>display activation status of display column <code>&lt;lpiterations&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.57 display/lpobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/lpobj/active</td>
<td>display activation status of display column <code>&lt;lpobj&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.58 display/maxdepth

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/maxdepth/active</td>
<td>display activation status of display column <code>&lt;maxdepth&gt;</code> (0: off, 1: auto, 2:on)</td>
<td>1 (0 for Windows without IDE)</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td></td>
</tr>
</tbody>
</table>
### 3.59 `display/memused`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/memused/active</code></td>
<td>display activation status of display column <code>&lt;memused&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.60 `display/nexternbranchcands`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/nexternbranchcands/active</code></td>
<td>display activation status of display column <code>&lt;nexternbranchcands&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1 (2 for nonlinear instances)</td>
</tr>
</tbody>
</table>

### 3.61 `display/nfrac`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/nfrac/active</code></td>
<td>display activation status of display column <code>&lt;nfrac&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1 (2 if discrete variables)</td>
</tr>
</tbody>
</table>

### 3.62 `display/nnodes`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/nnodes/active</code></td>
<td>display activation status of display column <code>&lt;nnodes&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.63 `display/nodesleft`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/nodesleft/active</code></td>
<td>display activation status of display column <code>&lt;nodesleft&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.64 `display/nsols`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display/nsols/active</code></td>
<td>display activation status of display column <code>&lt;nsols&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.65 `display/plungedepth`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>display/plungedepth/active</td>
<td>display activation status of display column <code>&lt;plungedepth&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.66 display/pools

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/poolsize/active</td>
<td>display activation status of display column <code>&lt;poolsize&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.67 display/primalbound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/primalbound/active</td>
<td>display activation status of display column <code>&lt;primalbound&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.68 display/primalgap

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/primalgap/active</td>
<td>display activation status of display column <code>&lt;primalgap&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.69 display/pseudoobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/pseudoobj/active</td>
<td>display activation status of display column <code>&lt;pseudoobj&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.70 display/separounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/separounds/active</td>
<td>display activation status of display column <code>&lt;separounds&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.71 display/solfound

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/solfound/active</td>
<td>display activation status of display column <code>&lt;solfound&gt;</code> (0: off, 1: auto, 2:on) Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.72 display/sols
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/sols/active</td>
<td>display activation status of display column <code>&lt;sols&gt;</code> (0: off, 1: auto, 2: on)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.73 display/strongbranches

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/strongbranches/active</td>
<td>display activation status of display column <code>&lt;strongbranches&gt;</code> (0: off, 1: auto, 2: on)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.74 display/time

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/time/active</td>
<td>display activation status of display column <code>&lt;time&gt;</code> (0: off, 1: auto, 2: on)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(2 for Windows without IDE)</td>
<td></td>
</tr>
</tbody>
</table>

### 3.75 display/vars

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>display/vars/active</td>
<td>display activation status of display column <code>&lt;vars&gt;</code> (0: off, 1: auto, 2: on)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 2]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.76 heuristics/actconsdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/actconsdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/actconsdiving/freq</td>
<td>frequency for calling primal heuristic <code>&lt;actconsdiving&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/actconsdiving/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;actconsdiving&gt;</code></td>
<td>5</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;actconsdiving&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/actconsdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Range: [0, 1]
### 3.77 heuristics/clique

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/clique/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/clique/freq</td>
<td>frequency for calling primal heuristic (&lt;clique&gt;) (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/clique/freqofs</td>
<td>frequency offset for calling primal heuristic (&lt;clique&gt;)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/clique/initseed</td>
<td>initial random seed value to permutate variables</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/clique/maxdepth</td>
<td>maximal depth level to call primal heuristic (&lt;clique&gt;) (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/clique/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/clique/maxproprounds</td>
<td>maximum number of propagation rounds during probing (-1 infinity)</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/clique/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixable</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/clique/minimprove</td>
<td>factor by which clique heuristic should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/clique/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/clique/multiplier</td>
<td>value to increase nodenumber to determine the next run</td>
<td>1.1</td>
</tr>
<tr>
<td>heuristics/clique/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/clique/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/clique/priority</td>
<td>priority of heuristic (&lt;clique&gt;)</td>
<td>-1000500</td>
</tr>
</tbody>
</table>

### 3.78 heuristics/coefdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/coefdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/coefdiving/freq</td>
<td>frequency for calling primal heuristic (&lt;coefdiving&gt;) (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/coefdiving/freqofs</td>
<td>frequency offset for calling primal heuristic (&lt;coefdiving&gt;)</td>
<td>1</td>
</tr>
</tbody>
</table>
### heuristics/coefdiving

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;coefdiving&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>maxdiveavgquot</td>
<td>maximal quotient (currlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>maxdiveubquot</td>
<td>maximal quotient (currlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td>maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td>priority of heuristic <code>&lt;coefdiving&gt;</code></td>
<td>-1001000</td>
</tr>
</tbody>
</table>

### heuristics/crossover

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>copycuts</td>
<td>if useprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>dontwaitatroot</td>
<td>should the nwaitingnodes parameter be ignored at the root node?</td>
<td>0</td>
</tr>
<tr>
<td>freq</td>
<td>frequency for calling primal heuristic <code>&lt;crossover&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>30</td>
</tr>
<tr>
<td>freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;crossover&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>limfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>2</td>
</tr>
<tr>
<td>maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;crossover&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixed</td>
<td>0.666</td>
</tr>
<tr>
<td>minimprove</td>
<td>factor by which Crossover should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td>minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td>nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td>nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td>nusedsols</td>
<td>number of solutions to be taken into account</td>
<td>3</td>
</tr>
<tr>
<td>nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
</tbody>
</table>
3.80 heuristics/dins

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dins/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/dins/freq</td>
<td>frequency for calling primal heuristic &lt;dins&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/dins/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;dins&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dins/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>1.5</td>
</tr>
<tr>
<td>heuristics/dins/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;dins&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/dins/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/dins/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixable</td>
<td>0.3</td>
</tr>
<tr>
<td>heuristics/dins/minimprove</td>
<td>factor by which dins should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/dins/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/dins/neighborhoodsize</td>
<td>radius (using Manhattan metric) of the incumbent’s neighborhood to be searched</td>
<td>18</td>
</tr>
<tr>
<td>heuristics/dins/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/dins/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/dins/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dins/priority</td>
<td>priority of heuristic &lt;dins&gt; Range: [-536870912, 536870911]</td>
<td>-1105000</td>
</tr>
<tr>
<td>heuristics/dins/solnum</td>
<td>number of pool-solutions to be checked for flag array update (for hard fixing of binary variables)</td>
<td>5</td>
</tr>
<tr>
<td>heuristics/dins/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
</tbody>
</table>

3.81 heuristics/dualval

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dualval/dynamicdepth</td>
<td>says if and how the recursion depth is computed at runtime Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/forceimprovements</td>
<td>exit if objective doesn’t improve</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/freq</td>
<td>frequency for calling primal heuristic &lt;dualval&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/dualval/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;dualval&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/heurverblevel</td>
<td>verblevel of the heuristic, default is 0 to display nothing Range: [0, 4]</td>
<td>0</td>
</tr>
</tbody>
</table>
### heuristics/dualval

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/dualval/lambdaobj</td>
<td>scaling factor for the objective function</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/dualval/lambdaobj</td>
<td>value added to objective of slack variables, must not be zero</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/dualval/maxcalls</td>
<td>maximal number of recursive calls of the heuristic (if dynamicdepth is off)</td>
<td>25</td>
</tr>
<tr>
<td>heuristics/dualval/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt; dualval &gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/dualval/maxequalranks</td>
<td>maximal number of variables that may have maximal rank, quit if there are more, turn off by setting -1</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/dualval/mingap</td>
<td>minimal gap for which we still run the heuristic, if gap is less we return without doing anything</td>
<td>5</td>
</tr>
<tr>
<td>heuristics/dualval/nlpverblevel</td>
<td>verblevel of the nlp solver, can be 0 or 1</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/nlpverblevel</td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/dualval/onlycheaper</td>
<td>add constraint to ensure that discrete vars are improving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/dualval/onlyleaves</td>
<td>disable the heuristic if it was not called at a leaf of the B&amp;B tree</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/priority</td>
<td>priority of heuristic &lt; dualval &gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/rankvalue</td>
<td>number of ranks that should be displayed when the heuristic is called</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/dualval/relaxcontvars</td>
<td>relax the continuous variables</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/dualval/relaxindicators</td>
<td>relax the indicator variables by introducing continuous copies</td>
<td>0</td>
</tr>
</tbody>
</table>

### heuristics/feaspump

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/feaspump/alpha</td>
<td>initial weight of the objective function in the convex combination</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/feaspump/alphadiff</td>
<td>threshold difference for the convex parameter to perform perturbation</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/feaspump/beforecuts</td>
<td>should the feasibility pump be called at root node before cut separation?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/feaspump/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/feaspump/cyclelength</td>
<td>maximum length of cycles to be checked explicitly in each round</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 100]</td>
<td></td>
</tr>
<tr>
<td>heuristics/feaspump/freq</td>
<td>frequency for calling primal heuristic &lt; feaspump &gt; (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td>heuristics/feaspump/freqofs</td>
<td>frequency offset for calling primal heuristic &lt; feaspump &gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/feaspump/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt; feaspump &gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/feaspump/maxloops</td>
<td>maximal number of pumping loops (-1: no limit)</td>
<td>10000</td>
</tr>
<tr>
<td>heuristics/feaspump/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/feaspump/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/feaspump/maxsols</td>
<td>total number of feasible solutions found up to which heuristic is called (-1: no limit)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/feaspump/maxstallloops</td>
<td>maximal number of pumping rounds without fractionality improvement (-1: no limit)</td>
<td>10</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>heuristics/feaspump/minflips</td>
<td>minimum number of random variables to flip, if a 1-cycle is encountered</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/feaspump/neighborhoodsize</td>
<td>radius (using Manhattan metric) of the neighborhood to be searched in stage 3</td>
<td>18</td>
</tr>
<tr>
<td>heuristics/feaspump/objfactor</td>
<td>factor by which the regard of the objective is decreased in each round, 1.0 for dynamic Range: [0, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/feaspump/pertsoffound</td>
<td>should a random perturbation be performed if a feasible solution was found?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/feaspump/perturbfreq</td>
<td>number of iterations until a random perturbation is forced</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/feaspump/priority</td>
<td>priority of heuristic &lt;feaspump&gt; Range: [-536870912, 536870911]</td>
<td>-1000000</td>
</tr>
<tr>
<td>heuristics/feaspump/stage3</td>
<td>should we solve a local branching sub-MIP if no solution could be found?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/feaspump/usefp20</td>
<td>should an iterative round-and-propagate scheme be used to find the integral points?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.83 heuristics/fixandinfer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/fixandinfer/freq</td>
<td>frequency for calling primal heuristic &lt;fixandinfer&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/fixandinfer/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;fixandinfer&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/fixandinfer/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;fixandinfer&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/fixandinfer/minfixings</td>
<td>minimal number of fixings to apply before dive may be aborted</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/fixandinfer/priority</td>
<td>priority of heuristic &lt;fixandinfer&gt; Range: [-536870912, 536870911]</td>
<td>-500000</td>
</tr>
<tr>
<td>heuristics/fixandinfer/proprounds</td>
<td>maximal number of propagation rounds in probing subproblems (-1: no limit, 0: auto)</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.84 heuristics/fracdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/fracdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/fracdiving/freq</td>
<td>frequency for calling primal heuristic &lt;fracdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/fracdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;fracdiving&gt;</td>
<td>3</td>
</tr>
<tr>
<td>heuristics/fracdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;fracdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/fracdiving/maxdiveavgquot</td>
<td>maximal quotient (crlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/fracdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/fracdiving/maxdiveubquot</td>
<td>maximal quotient (crlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]</td>
<td>0.8</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>heuristics/fractiondiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fractiondiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/fractiondiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/fractiondiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fractiondiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/fractiondiving/priority</td>
<td>priority of heuristic &lt;fractiondiving&gt;</td>
<td>-1003000</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.85 heuristics/guideddiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/guideddiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/guideddiving/freq</td>
<td>frequency for calling primal heuristic &lt;guideddiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/guideddiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;guideddiving&gt;</td>
<td>7</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;guideddiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/guideddiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/guideddiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/guideddiving/priority</td>
<td>priority of heuristic &lt;guideddiving&gt;</td>
<td>-1007000</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.86 heuristics/intdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/intdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/intdiving/freq</td>
<td>frequency for calling primal heuristic &lt;intdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/intdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;intdiving&gt;</td>
<td>9</td>
</tr>
<tr>
<td>heuristics/intdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;intdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveavgquot</td>
<td>maximal quotient ( (\text{curlowerbound} - \text{lowerbound}) / (\text{avglowerbound} - \text{lowerbound}) ) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveubquot</td>
<td>maximal quotient ( (\text{curlowerbound} - \text{lowerbound}) / (\text{cutoffbound} - \text{lowerbound}) ) where diving is performed (0.0: no limit) Range: ([0, 1])</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/intdiving/maxdiveubquotnosol</td>
<td>maximal UBUQUOT when no solution was found yet (0.0: no limit) Range: ([0, 1])</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/intdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/intdiving/maxlpiterofs</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/intdiving/maxreldepth</td>
<td>maximal relative depth to start diving Range: ([0, 1])</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/intdiving/minreldepth</td>
<td>minimal relative depth to start diving Range: ([0, 1])</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/intdiving/priority</td>
<td>priority of heuristic (&lt;\text{intdiving}&gt;) Range: ([-536870912, 536870911])</td>
<td>-1003500</td>
</tr>
</tbody>
</table>

### 3.87 heuristics/intshifting

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/intshifting/freq</td>
<td>frequency for calling primal heuristic (&lt;\text{intshifting})&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/intshifting/freqofs</td>
<td>frequency offset for calling primal heuristic (&lt;\text{intshifting}&gt;)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/intshifting/maxdepth</td>
<td>maximal depth level to call primal heuristic (&lt;\text{intshifting}&gt;) (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/intshifting/priority</td>
<td>priority of heuristic (&lt;\text{intshifting}&gt;) Range: ([-536870912, 536870911])</td>
<td>-10000</td>
</tr>
</tbody>
</table>

### 3.88 heuristics/linesearchdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/linesearchdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/freq</td>
<td>frequency for calling primal heuristic (&lt;\text{linesearchdiving}&gt;) (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/freqofs</td>
<td>frequency offset for calling primal heuristic (&lt;\text{linesearchdiving}&gt;)</td>
<td>6</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic (&lt;\text{linesearchdiving}&gt;) (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxdiveavgquot</td>
<td>maximal quotient ( (\text{curlowerbound} - \text{lowerbound}) / (\text{avglowerbound} - \text{lowerbound}) ) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/linesearchdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
</tbody>
</table>
heuristics/linsearchdiving/maxdiveubquot | maximal quotient (curobjective - lowerbound)/(cutoffbound - lowerbound) when diving is performed (0.0: no limit) | 0.8 |
-------------------------------------------------|-------------------------------------------------|---------------------|
heuristics/linsearchdiving/maxdiveubquotnosol | maximal UBQUOT when no solution was found yet (0.0: no limit) | 0.1 |
heuristics/linsearchdiving/maxlpiterofs | additional number of allowed LP iterations | 1000 |
heuristics/linsearchdiving/maxlpiterquot | maximal fraction of diving LP iterations compared to node LP iterations | 0.05 |
heuristics/linsearchdiving/maxreldepth | maximal relative depth to start diving | 1 |
heuristics/linsearchdiving/minreldepth | minimal relative depth to start diving | 0 |
heuristics/linsearchdiving/priority | priority of heuristic <linsearchdiving> | -1006000 |

3.89 heuristics/localbranching

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/localbranching/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/localbranching/freq</td>
<td>frequency for calling primal heuristic &lt;localbranching&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/localbranching/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;localbranching&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/localbranching/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>1.5</td>
</tr>
<tr>
<td>heuristics/localbranching/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;localbranching&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/localbranching/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>10000</td>
</tr>
<tr>
<td>heuristics/localbranching/minimprove</td>
<td>factor by which localbranching should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/localbranching/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/localbranching/neighborhoodsize</td>
<td>radius (using Manhattan metric) of the incumbent’s neighborhood to be searched</td>
<td>18</td>
</tr>
<tr>
<td>heuristics/localbranching/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/localbranching/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/localbranching/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
<tr>
<td>heuristics/localbranching/priority</td>
<td>priority of heuristic &lt;localbranching&gt;</td>
<td>-1102000</td>
</tr>
<tr>
<td>heuristics/localbranching/uselprows</td>
<td>should subproblem be created out of the rows of the LP rows?</td>
<td>0</td>
</tr>
</tbody>
</table>
### 3.90 heuristics/mutation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/mutation/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/mutation/freq</td>
<td>frequency for calling primal heuristic &lt;mutation&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/mutation/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;mutation&gt;</td>
<td>8</td>
</tr>
<tr>
<td>heuristics/mutation/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;mutation&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/mutation/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/mutation/minfixingrate</td>
<td>percentage of integer variables that have to be fixed</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/mutation/minimprove</td>
<td>factor by which mutation should at least improve the incumbent Range: [1e-06, 0.999999]</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/mutation/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/mutation/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/mutation/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/mutation/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
<tr>
<td>heuristics/mutation/priority</td>
<td>priority of heuristic &lt;mutation&gt; Range: [-536870912, 536870911]</td>
<td>-1103000</td>
</tr>
<tr>
<td>heuristics/mutation/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.91 heuristics/nlpdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/nlpdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/nlpdiving/fixquot</td>
<td>percentage of fractional variables that should be fixed before the next NLP solve Range: [0, 1]</td>
<td>0.2</td>
</tr>
<tr>
<td>heuristics/nlpdiving/freq</td>
<td>frequency for calling primal heuristic &lt;nlpdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/nlpdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;nlpdiving&gt;</td>
<td>3</td>
</tr>
<tr>
<td>heuristics/nlpdiving/lp</td>
<td>should the LP relaxation be solved before the NLP relaxation?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;nlpdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/nlpdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]</td>
<td>0.8</td>
</tr>
</tbody>
</table>
3.92 heuristics/objpscostdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/objpscostdiving/depthfac</td>
<td>maximal diving depth: number of binary/integer variables times depthfac</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/depthfacnosol</td>
<td>maximal diving depth factor if no feasible solution was found yet</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/freq</td>
<td>frequency for calling primal heuristic &lt;objpscostdiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;objpscostdiving&gt;</td>
<td>4</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;objpscostdiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to total iteration number</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxreldepth</td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/maxsols</td>
<td>total number of feasible solutions found up to which heuristic is called (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/objpscostdiving/minreldepth</td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
</tbody>
</table>
### 3.93 heuristics/octane

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/octane/ffirst</td>
<td>number of 0-1-points to be tested at first whether they violate a common row</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/octane/fmax</td>
<td>number of 0-1-points to be tested as possible solutions by OCTANE</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/octane/freq</td>
<td>frequency for calling primal heuristic &lt;octane&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/octane/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;octane&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/octane/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;octane&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/octane/priority</td>
<td>priority of heuristic &lt;octane&gt;</td>
<td>-1008000</td>
</tr>
<tr>
<td>heuristics/octane/useavgnbray</td>
<td>should the weighted average of the nonbasic cone be used as one ray direction?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/useavgray</td>
<td>should the average of the basic cone be used as one ray direction?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/useavgwgtray</td>
<td>should the weighted average of the basic cone be used as one ray direction?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/usediffray</td>
<td>should the difference between the root solution and the current LP solution be used as one ray direction?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/octane/usefracspace</td>
<td>execute OCTANE only in the space of fractional variables (TRUE) or in the full space?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/octane/useobjray</td>
<td>should the inner normal of the objective be used as one ray direction?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.94 heuristics/oneopt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/oneopt/beforepresol</td>
<td>should the heuristic be called before presolving?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/oneopt/duringroot</td>
<td>should the heuristic be called before and during the root node?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/oneopt/forcelpconstruction</td>
<td>should the construction of the LP be forced even if LP solving is deactivated?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/oneopt/freq</td>
<td>frequency for calling primal heuristic &lt;oneopt&gt; (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/oneopt/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;oneopt&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/oneopt/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;oneopt&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/oneopt/priority</td>
<td>priority of heuristic &lt;oneopt&gt;</td>
<td>-20000</td>
</tr>
<tr>
<td>heuristics/oneopt/weightedobj</td>
<td>should the objective be weighted with the potential shifting value when sorting the shifting candidates?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.95 heuristics/proximity

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/proximity/binvarquot</td>
<td>threshold for percentage of binary variables required to start</td>
<td>0.1</td>
</tr>
</tbody>
</table>
### heuristics/proximity

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/proximity/freq</td>
<td>frequency for calling primal heuristic <code>&lt;proximity&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/proximity/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;proximity&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/proximity/lpitersquot</td>
<td>quotient of sub-MIP LP iterations with respect to LP iterations so far</td>
<td>0.2</td>
</tr>
<tr>
<td>heuristics/proximity/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;proximity&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/proximity/maxlpiters</td>
<td>maximum number of LP iterations to be performed in the subproblem</td>
<td>100000</td>
</tr>
<tr>
<td>heuristics/proximity/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>10000</td>
</tr>
<tr>
<td>heuristics/proximity/mingap</td>
<td>minimum primal-dual gap for which the heuristic is executed</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/proximity/minimprove</td>
<td>factor by which proximity should at least improve the incumbent</td>
<td>0.02</td>
</tr>
<tr>
<td>heuristics/proximity/minlpiters</td>
<td>minimum number of LP iterations performed in subproblem</td>
<td>200</td>
</tr>
<tr>
<td>heuristics/proximity/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/proximity/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/proximity/nodesquot</td>
<td>sub-MIP node limit w.r.t number of original nodes</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/proximity/priority</td>
<td>priority of heuristic <code>&lt;proximity&gt;</code></td>
<td>-200000</td>
</tr>
<tr>
<td>heuristics/proximity/restart</td>
<td>should the heuristic immediately run again on its newly found solution?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/proximity/usefinallp</td>
<td>should the heuristic solve a final LP in case of continuous objective variables?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/proximity/uselprows</td>
<td>should subproblem be constructed based on LP row information?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/proximity/waitingnodes</td>
<td>waiting nodes since last incumbent before heuristic is executed</td>
<td>100</td>
</tr>
</tbody>
</table>

### 3.96 heuristics/pscostdiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/pscostdiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/pscostdiving/freq</td>
<td>frequency for calling primal heuristic <code>&lt;pscostdiving&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/pscostdiving/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;pscostdiving&gt;</code></td>
<td>2</td>
</tr>
<tr>
<td>heuristics/pscostdiving/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;pscostdiving&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/pscostdiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/pscostdiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/pscostdiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/pscostdiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit)</td>
<td>0.1</td>
</tr>
</tbody>
</table>
### 3.97 heuristics/randrounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/randrounding/freq</td>
<td>frequency for calling primal heuristic <code>&lt;randrounding&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td>heuristics/randrounding/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;randrounding&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/randrounding/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;randrounding&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/randrounding/maxpropronds</td>
<td>limit of rounds for each propagation call</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/randrounding/oncepernode</td>
<td>should the heuristic only be called once per node?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/randrounding/priority</td>
<td>priority of heuristic <code>&lt;randrounding&gt;</code></td>
<td>-200</td>
</tr>
<tr>
<td>heuristics/randrounding/propagately</td>
<td>should the probing part of the heuristic be applied exclusively at the root node?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/randrounding/usesimplerounding</td>
<td>should the heuristic apply the variable lock strategy of simple rounding, if possible?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.98 heuristics/rens

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rens/addallsols</td>
<td>should all subproblem solutions be added to the original SCIP?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rens/binarybounds</td>
<td>should general integers get binary bounds [floor(,),ceil(,)] ?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/rens/copycuts</td>
<td>if usebox == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/rens/extratime</td>
<td>should the RENS sub-CIP get its own full time limit? This is only for testing and not recommended!</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rens/freq</td>
<td>frequency for calling primal heuristic <code>&lt;rens&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rens/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;rens&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rens/fullscale</td>
<td>should the RENS sub-CIP be solved with cuts, conflicts, strong branching,...</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>This is only for testing and not recommended!</td>
<td></td>
</tr>
<tr>
<td>heuristics/rens/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/rens/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;rens&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/rens/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/rens/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixable</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
</tbody>
</table>
heuristics/rens/minimprove | factor by which RENS should at least improve the incumbent | 0.01
heuristics/rens/minnodes | minimum number of nodes required to start the subproblem | 50
heuristics/rens/nodesofs | number of nodes added to the contingent of the total nodes | 500
heuristics/rens/nodesquot | contingent of sub problem nodes in relation to the number of nodes of the original problem | 0.1
heuristics/rens/priority | priority of heuristic <rens> | -1100000
heuristics/rens/startsol | solution that is used for fixing values ('l'p relaxation, 'n’lp relaxation) | 1
heuristics/rens/uselprows | should subproblem be created out of the rows in the LP rows? | 0

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rens/copycuts</td>
<td>if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/rens/freq</td>
<td>frequency for calling primal heuristic &lt;rens&gt; (-1: never, 0: only at depth freqofs)</td>
<td>25</td>
</tr>
<tr>
<td>heuristics/rens/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;rens&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/rens/lplimfac</td>
<td>factor by which the limit on the number of LP depends on the node limit</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/rens/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;rens&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/rens/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/rens/minfixingrate</td>
<td>minimum percentage of integer variables that have to be fixed</td>
<td>0.3</td>
</tr>
<tr>
<td>heuristics/rens/minimprove</td>
<td>factor by which rins should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/rens/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>50</td>
</tr>
<tr>
<td>heuristics/rens/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/rens/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/rens/nwaitingnodes</td>
<td>number of nodes without incumbent change that heuristic should wait</td>
<td>200</td>
</tr>
<tr>
<td>heuristics/rens/priority</td>
<td>priority of heuristic &lt;rens&gt;</td>
<td>-1101000</td>
</tr>
<tr>
<td>heuristics/rens/uselprows</td>
<td>should subproblem be created out of the rows in the LP rows?</td>
<td>0</td>
</tr>
</tbody>
</table>

3.100 heuristics/rootsoldiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/rootsoldiving/alpha</td>
<td>soft rounding factor to fade out objective coefficients</td>
<td>0.9</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/depthfac</td>
<td>maximal diving depth: number of binary/integer variables times depthfac</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/depthfacnosol</td>
<td>maximal diving depth factor if no feasible solution was found yet</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/rootsoldiving/freq</td>
<td>frequency for calling primal heuristic &lt;rootsoldiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>20</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/freqofs</code></td>
<td>frequency offset for calling primal heuristic <code>&lt;rootsoldiving&gt;</code></td>
<td>5</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/maxdepth</code></td>
<td>maximal depth level to call primal heuristic <code>&lt;rootsoldiving&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/maxlpiterofs</code></td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/maxlpiterquot</code></td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.01</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/maxreldepth</code></td>
<td>maximal relative depth to start diving</td>
<td>1</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/maxsols</code></td>
<td>total number of feasible solutions found up to which heuristic is called (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/minreldepth</code></td>
<td>minimal relative depth to start diving</td>
<td>0</td>
</tr>
<tr>
<td><code>heuristics/rootsoldiving/priority</code></td>
<td>priority of heuristic <code>&lt;rootsoldiving&gt;</code></td>
<td>-1005000</td>
</tr>
</tbody>
</table>

### 3.101 heuristics/rounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heuristics/rounding/freq</code></td>
<td>frequency for calling primal heuristic <code>&lt;rounding&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td><code>heuristics/rounding/freqofs</code></td>
<td>frequency offset for calling primal heuristic <code>&lt;rounding&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td><code>heuristics/rounding/maxdepth</code></td>
<td>maximal depth level to call primal heuristic <code>&lt;rounding&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><code>heuristics/rounding/oncepernode</code></td>
<td>should the heuristic only be called once per node?</td>
<td>0</td>
</tr>
<tr>
<td><code>heuristics/rounding/priority</code></td>
<td>priority of heuristic <code>&lt;rounding&gt;</code></td>
<td>-1000</td>
</tr>
<tr>
<td><code>heuristics/rounding/successfactor</code></td>
<td>number of calls per found solution that are considered as standard success, a higher factor causes the heuristic to be called more often</td>
<td>100</td>
</tr>
</tbody>
</table>

### 3.102 heuristics/shiftandpropagate

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heuristics/shiftandpropagate/collectstats</code></td>
<td>should variable statistics be collected during probing?</td>
<td>1</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/cutoffbreaker</code></td>
<td>The number of cutoffs before heuristic stops</td>
<td>15</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/fixbinlocks</code></td>
<td>should binary variables with no locks in one direction be fixed to that direction?</td>
<td>1</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/freq</code></td>
<td>frequency for calling primal heuristic <code>&lt;shiftandpropagate&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/freqofs</code></td>
<td>frequency offset for calling primal heuristic <code>&lt;shiftandpropagate&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/impliscontinuous</code></td>
<td>should implicit integer variables be treated as continuous variables?</td>
<td>1</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/maxdepth</code></td>
<td>maximal depth level to call primal heuristic <code>&lt;shiftandpropagate&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/normalize</code></td>
<td>should coefficients and left/right hand sides be normalized by max row coeff?</td>
<td>1</td>
</tr>
<tr>
<td><code>heuristics/shiftandpropagate/nozerofixing</code></td>
<td>should variables with a zero shifting value be delayed instead of being fixed?</td>
<td>0</td>
</tr>
</tbody>
</table>
### 3.103 heuristics/shifting

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/shifting/freq</td>
<td>frequency for calling primal heuristic &lt;shifting&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/shifting/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;shifting&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/shifting/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;shifting&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/shifting/priority</td>
<td>priority of heuristic &lt;shifting&gt;</td>
<td>-5000</td>
</tr>
</tbody>
</table>

### 3.104 heuristics/simplerounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/simplerounding/freq</td>
<td>frequency for calling primal heuristic &lt;simplerounding&gt; (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/simplerounding/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;simplerounding&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/simplerounding/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;simplerounding&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/simplerounding/oncepernode</td>
<td>should the heuristic only be called once per node?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/simplerounding/priority</td>
<td>priority of heuristic &lt;simplerounding&gt;</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.105 heuristics/subnlp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/subnlp/forbidfixings</td>
<td>whether to add constraints that forbid specific fixings that turned out to be infeasible</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>heuristics/subnlp/freq</td>
<td>frequency for calling primal heuristic <code>&lt;subnlp&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/subnlp/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;subnlp&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/subnlp/itermin</td>
<td>contingent of NLP iterations in relation to the number of nodes in SCIP</td>
<td>300</td>
</tr>
<tr>
<td>heuristics/subnlp/iteroffset</td>
<td>number of iterations added to the contingent of the total number of iterations</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/subnlp/iterquotient</td>
<td>contingent of NLP iterations in relation to the number of nodes in SCIP</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/subnlp/keepcopy</td>
<td>whether to keep SCIP copy or to create new copy each time heuristic is applied</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/subnlp/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;subnlp&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/subnlp/maxpresolverounds</td>
<td>limit on number of presolve rounds in sub-SCIP (-1 for unlimited, 0 for no presolve)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/subnlp/minimprove</td>
<td>factor by which NLP heuristic should at least improve the incumbent Range: [0, 1]</td>
<td>0.01</td>
</tr>
<tr>
<td>heuristics/subnlp/nlpiterlimit</td>
<td>iteration limit of NLP solver; 0 to use solver default</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/subnlp/nlpoptfile</td>
<td>name of an NLP solver specific options file</td>
<td></td>
</tr>
<tr>
<td>heuristics/subnlp/nlp timelimit</td>
<td>time limit of NLP solver; 0 to use solver default</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/subnlp/nlpverblevel</td>
<td>verbosity level of NLP solver</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/subnlp/priority</td>
<td>priority of heuristic <code>&lt;subnlp&gt;</code> Range: [-536870912, 536870911]</td>
<td>-2000000</td>
</tr>
<tr>
<td>heuristics/subnlp/resolvetol</td>
<td>should the NLP resolve be started from the original starting point or the infeasible solution?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/subnlp/resolvetolfactor</td>
<td>if SCIP does not accept a NLP feasible solution, resolve NLP with feas. tolerance reduced by this factor (set to 1.0 to turn off resolve) Range: [0, 1]</td>
<td>0.001</td>
</tr>
<tr>
<td>heuristics/subnlp/runalways</td>
<td>whether to run NLP heuristic always if starting point available (does not use iteroffset,iterquot,itermin)</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.106 heuristics/trivial

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trivial/freq</td>
<td>frequency for calling primal heuristic <code>&lt;trivial&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/trivial/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;trivial&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/trivial/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;trivial&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/trivial/priority</td>
<td>priority of heuristic <code>&lt;trivial&gt;</code> Range: [-536870912, 536870911]</td>
<td>10000</td>
</tr>
</tbody>
</table>

### 3.107 heuristics/trysol

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/trysol/freq</td>
<td>frequency for calling primal heuristic <code>&lt;trysol&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/trysol/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;trysol&gt;</code></td>
<td>0</td>
</tr>
</tbody>
</table>
## 3.108 heuristics/twoopt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/twoopt/freq</td>
<td>frequency for calling primal heuristic <code>&lt;twoopt&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/twoopt/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;twoopt&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/twoopt/intopt</td>
<td>Should Integer-2-Optimization be applied or not?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/twoopt/matchingrate</td>
<td>parameter to determine the percentage of rows two variables have to share before they are considered equal Range: [0, 1]</td>
<td>0.5</td>
</tr>
<tr>
<td>heuristics/twoopt/maxdepth</td>
<td>maximal depth level to call primal heuristic <code>&lt;twoopt&gt;</code> (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/twoopt/maxnslaves</td>
<td>maximum number of slaves for one master variable Range: [-1, 1000000]</td>
<td>199</td>
</tr>
<tr>
<td>heuristics/twoopt/priority</td>
<td>priority of heuristic <code>&lt;twoopt&gt;</code> Range: [-536870912, 536870911]</td>
<td>-20100</td>
</tr>
<tr>
<td>heuristics/twoopt/waitingnodes</td>
<td>user parameter to determine number of nodes to wait after last best solution before calling heuristic Range: [0, 10000]</td>
<td>0</td>
</tr>
</tbody>
</table>

## 3.109 heuristics/undercover

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/undercover/beforecuts</td>
<td>should the heuristic be called at root node before cut separation?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/undercover/conflictweight</td>
<td>weight for conflict score in fixing order</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/undercover/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/undercover/coverbd</td>
<td>should bounddisjunction constraints be covered (or just copied)?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/coveringobj</td>
<td>objective function of the covering problem (influenced nonlinear constraints/terminology, domain size, locks, min of up/down locks, unit penalties)</td>
<td>u</td>
</tr>
<tr>
<td>heuristics/undercover/cutoffweight</td>
<td>weight for cutoff score in fixing order</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/undercover/fixingalts</td>
<td>prioritized sequence of fixing values used ('lp relaxation, incumbent solution)</td>
<td>li</td>
</tr>
<tr>
<td>heuristics/undercover/fixingorder</td>
<td>order in which variables should be fixed (increasing conflict score, decreasing conflict score, increasing variable index, decreasing variable index)</td>
<td>v</td>
</tr>
<tr>
<td>heuristics/undercover/fixintfirst</td>
<td>should integer variables in the cover be fixed first?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/freq</td>
<td>frequency for calling primal heuristic <code>&lt;undercover&gt;</code> (-1: never, 0: only at depth freqofs)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/freqofs</td>
<td>frequency offset for calling primal heuristic <code>&lt;undercover&gt;</code></td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/inferenceweight</td>
<td>weight for inference score in fixing order</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>heuristics/undercover/locksrounding</td>
<td>shall LP values for integer vars be rounded according to locks?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/undercover/maxbacktracks</td>
<td>maximum number of backtracks in fix-and-propagate</td>
<td>6</td>
</tr>
<tr>
<td>heuristics/undercover/maxcoversizeconss</td>
<td>maximum coversize maximum coversize (as ratio to the percentage of non-affected constraints)</td>
<td>maxdouble</td>
</tr>
<tr>
<td>heuristics/undercover/maxcoversizevars</td>
<td>maximum coversize (as fraction of total number of variables) Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/undercover/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;undercover&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/undercover/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/undercover/maxreorders</td>
<td>maximum number of reorderings of the fixing order</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/undercover/mincoveredabs</td>
<td>minimum number of nonlinear constraints in the original problem</td>
<td>5</td>
</tr>
<tr>
<td>heuristics/undercover/mincoveredrel</td>
<td>minimum percentage of nonlinear constraints in the original problem Range: [0, 1]</td>
<td>0.15</td>
</tr>
<tr>
<td>heuristics/undercover/minimprove</td>
<td>factor by which the heuristic should at least improve the incumbent Range: [-1, 1]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/undercover/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>500</td>
</tr>
<tr>
<td>heuristics/undercover/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/undercover/onlyconvexify</td>
<td>should we only fix variables in order to obtain a convex problem?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/undercover/postnlp</td>
<td>should the NLP heuristic be called to polish a feasible solution?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/undercover/priority</td>
<td>priority of heuristic &lt;undercover&gt; Range: [-536870912, 536870911]</td>
<td>-1110000</td>
</tr>
<tr>
<td>heuristics/undercover/recoverdiv</td>
<td>fraction of covering variables in the last cover which need to change their value when recovering Range: [0, 1]</td>
<td>0.9</td>
</tr>
<tr>
<td>heuristics/undercover/reusecover</td>
<td>shall the cover be reused if a conflict was added after an infeasible subproblem?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.110  heuristics/vbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/vbounds/copycuts</td>
<td>should all active cuts from cutpool be copied to constraints in subproblem?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/vbounds/freq</td>
<td>frequency for calling primal heuristic &lt;vbounds&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/vbounds/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;vbounds&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/vbounds/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;vbounds&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/vbounds/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>5000</td>
</tr>
</tbody>
</table>
heuristics/vbounds/maxproprounds: maximum number of propagation rounds during probing (-1 infinity)
  Range: [-1, 536870911]  2

heuristics/vbounds/minfixingrate: minimum percentage of integer variables that have to be fixable
  Range: [0, 1]  0.5

heuristics/vbounds/minimprove: factor by which vbounds heuristic should at least improve the incumbent
  Range: [0, 1]  0.01

heuristics/vbounds/minnodes: minimum number of nodes required to start the subproblem
  500

heuristics/vbounds/nodesofs: number of nodes added to the contingent of the total nodes
  500

heuristics/vbounds/nodesquot: contingent of sub problem nodes in relation to the number of nodes of the original problem
  Range: [0, 1]  0.1

heuristics/vbounds/priority: priority of heuristic <vbounds>
  Range: [-536870912, 536870911]  -1106000

3.111 heuristics/veclendiving

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/veclendiving/backtrack</td>
<td>use one level of backtracking if infeasibility is encountered?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/veclendiving/freq</td>
<td>frequency for calling primal heuristic &lt;veclendiving&gt; (-1: never, 0: only at depth freqofs)</td>
<td>10</td>
</tr>
<tr>
<td>heuristics/veclendiving/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;veclendiving&gt;</td>
<td>4</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;veclendiving&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxdiveavgquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxdiveavgquotnosol</td>
<td>maximal AVGQUOT when no solution was found yet (0.0: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxdiveubquot</td>
<td>maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]</td>
<td>0.8</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxdiveubquotnosol</td>
<td>maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]</td>
<td>0.1</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxlpiterofs</td>
<td>additional number of allowed LP iterations</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxlpiterquot</td>
<td>maximal fraction of diving LP iterations compared to node LP iterations</td>
<td>0.05</td>
</tr>
<tr>
<td>heuristics/veclendiving/maxreldepth</td>
<td>maximal relative depth to start diving Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/veclendiving/minreldepth</td>
<td>minimal relative depth to start diving Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/veclendiving/priority</td>
<td>priority of heuristic &lt;veclendiving&gt; Range: [-536870912, 536870911]</td>
<td>-1003100</td>
</tr>
</tbody>
</table>
### 3.112 heuristics/zeroobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/zeroobj/addallsols</td>
<td>should all subproblem solutions be added to the original SCIP?</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/zeroobj/freq</td>
<td>frequency for calling primal heuristic &lt;zeroobj&gt; (-1: never, 0: only at depth freqofs)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/zeroobj/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;zeroobj&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/zeroobj/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;zeroobj&gt; (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/zeroobj/maxlpiters</td>
<td>maximum number of LP iterations to be performed in the subproblem</td>
<td>5000</td>
</tr>
<tr>
<td>heuristics/zeroobj/maxnodes</td>
<td>maximum number of nodes to regard in the subproblem</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/zeroobj/minimprove</td>
<td>factor by which zeroobj should at least improve the incumbent</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/zeroobj/minnodes</td>
<td>minimum number of nodes required to start the subproblem</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/zeroobj/nodesofs</td>
<td>number of nodes added to the contingent of the total nodes</td>
<td>100</td>
</tr>
<tr>
<td>heuristics/zeroobj/nodesquot</td>
<td>contingent of sub problem nodes in relation to the number of nodes of the original problem</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/zeroobj/onlywithoutsol</td>
<td>should heuristic only be executed if no primal solution was found, yet?</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/zeroobj/priority</td>
<td>priority of heuristic &lt;zeroobj&gt;</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.113 heuristics/zirounding

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>heuristics/zirounding/freq</td>
<td>frequency for calling primal heuristic &lt;zirounding&gt; (-1: never, 0: only at depth freqofs)</td>
<td>1</td>
</tr>
<tr>
<td>heuristics/zirounding/freqofs</td>
<td>frequency offset for calling primal heuristic &lt;zirounding&gt;</td>
<td>0</td>
</tr>
<tr>
<td>heuristics/zirounding/maxdepth</td>
<td>maximal depth level to call primal heuristic &lt;zirounding&gt; (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>heuristics/zirounding/maxroundingloops</td>
<td>determines maximum number of rounding loops</td>
<td>2</td>
</tr>
<tr>
<td>heuristics/zirounding/minstopncalls</td>
<td>determines the minimum number of calls before percentage-based deactivation of Zirounding is applied</td>
<td>1000</td>
</tr>
<tr>
<td>heuristics/zirounding/priority</td>
<td>priority of heuristic &lt;zirounding&gt;</td>
<td>-500</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
<tr>
<td>heuristics/zirounding/stoppercentage</td>
<td>if percentage of found solutions falls below this parameter, Zirounding will be deactivated</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>heuristics/zirounding/stopziround</td>
<td>flag to determine if Zirounding is deactivated after a certain percentage of unsuccessful calls</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.114 history

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>history/valuebased</td>
<td>should statistics be collected for variable domain value pairs?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.115 limits
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>limits/absgap</td>
<td>solving stops, if the absolute gap =</td>
<td>primalbound - dualbound</td>
</tr>
<tr>
<td>limits/bestsol</td>
<td>solving stops, if the given number of solution improvements were found (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>limits/gap</td>
<td>solving stops, if the relative gap =</td>
<td>primal - dual</td>
</tr>
<tr>
<td>limits/maxorigsol</td>
<td>maximal number of solutions candidates to store in the solution storage of the original problem</td>
<td>10</td>
</tr>
<tr>
<td>limits/maxsol</td>
<td>maximal number of solutions to store in the solution storage</td>
<td>100</td>
</tr>
<tr>
<td>limits/memory</td>
<td>maximal memory usage in MB; reported memory usage is lower than real memory usage!</td>
<td>GAMS workspace</td>
</tr>
<tr>
<td>limits/nodes</td>
<td>maximal number of nodes to process (-1: no limit)</td>
<td>GAMS nodlim, if set, otherwise -1</td>
</tr>
<tr>
<td>limits/restarts</td>
<td>solving stops, if the given number of restarts was triggered (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>limits/solutions</td>
<td>solving stops, if the given number of solutions were found (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>limits/stallnodes</td>
<td>solving stops, if the given number of nodes was processed since the last improvement of the primal solution value (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>limits/time</td>
<td>maximal time in seconds to run</td>
<td>GAMS reslim</td>
</tr>
<tr>
<td>limits/totalnodes</td>
<td>maximal number of total nodes (incl. restarts) to process (-1: no limit)</td>
<td>-1</td>
</tr>
</tbody>
</table>

### 3.116 lp

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp/checkdualfeas</td>
<td>should LP solutions be checked for dual feasibility, resolving LP when numerical troubles occur?</td>
<td>1</td>
</tr>
<tr>
<td>lp/checkprimfeas</td>
<td>should LP solutions be checked for primal feasibility, resolving LP when numerical troubles occur?</td>
<td>1</td>
</tr>
<tr>
<td>lp/checkstability</td>
<td>should LP solver’s return status be checked for stability?</td>
<td>1</td>
</tr>
<tr>
<td>lp/cleanupcols</td>
<td>should new non-basic columns be removed after LP solving?</td>
<td>0</td>
</tr>
<tr>
<td>lp/cleanupcolsroot</td>
<td>should new non-basic columns be removed after root LP solving?</td>
<td>0</td>
</tr>
<tr>
<td>lp/cleanuprows</td>
<td>should new basic rows be removed after LP solving?</td>
<td>1</td>
</tr>
<tr>
<td>lp/cleanuprowsroot</td>
<td>should new basic rows be removed after root LP solving?</td>
<td>1</td>
</tr>
<tr>
<td>lp/clearinitialprobinglp</td>
<td>should lp state be cleared at the end of probing mode when lp was initially unsolved, e.g., when called right after presolving?</td>
<td>1</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>lp/colagelimit</td>
<td>Maximum age a dynamic column can reach before it is deleted from the LP (-1: don’t delete columns due to aging)</td>
<td>10</td>
</tr>
<tr>
<td>lp/conditionlimit</td>
<td>Maximum condition number of LP basis counted as stable (-1.0: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>lp/disablecutoff</td>
<td>Disable the cutoff bound in the LP solver? (0: enabled, 1: disabled, 2: auto)</td>
<td>2</td>
</tr>
<tr>
<td>lp/fastmip</td>
<td>Which FASTMIP setting of LP solver should be used? (0: off, 1: low)</td>
<td>1</td>
</tr>
<tr>
<td>lp/freesolvalbuffers</td>
<td>Should the buffers for storing LP solution values during diving be freed at end of diving?</td>
<td>0</td>
</tr>
<tr>
<td>lp/initalgorithm</td>
<td>LP algorithm for solving initial LP relaxations (automatic ’s’implex, ’p’rimal simplex, ’d’ual simplex, ’b’arrier, barrier with ’c’rossover)</td>
<td>s</td>
</tr>
<tr>
<td>lp/iterlim</td>
<td>Iteration limit for each single LP solve (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>lp/lexdualalgo</td>
<td>Should the lexicographic dual algorithm be used?</td>
<td>0</td>
</tr>
<tr>
<td>lp/lexdualbasic</td>
<td>Choose fractional basic variables in lexicographic dual algorithm?</td>
<td>0</td>
</tr>
<tr>
<td>lp/lexdualmaxrounds</td>
<td>Maximum number of rounds in the lexicographic dual algorithm (-1: unbounded)</td>
<td>2</td>
</tr>
<tr>
<td>lp/lexdualrootonly</td>
<td>Should the lexicographic dual algorithm be applied only at the root node</td>
<td>1</td>
</tr>
<tr>
<td>lp/lexdualstalling</td>
<td>Turn on the lex dual algorithm only when stalling?</td>
<td>1</td>
</tr>
<tr>
<td>lp/presolving</td>
<td>Should presolving of LP solver be used?</td>
<td>1</td>
</tr>
<tr>
<td>lp/pricing</td>
<td>LP pricing strategy (’l’pi default, ’a’uto, ’f’ull pricing, ’p’artial, ’s’teeepest edge pricing, ’q’ueeekstart steepest edge pricing, ’d’eex pricing)</td>
<td>1</td>
</tr>
<tr>
<td>lp/resolvealgorithm</td>
<td>LP algorithm for resolving LP relaxations if a starting basis exists (automatic ’s’implex, ’p’rimal simplex, ’d’ual simplex, ’b’arrier, barrier with ’c’rossover)</td>
<td>s</td>
</tr>
<tr>
<td>lp/resolveiterfac</td>
<td>Factor of average LP iterations that is used as LP iteration limit for LP resolve (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>lp/resolveitermin</td>
<td>Minimum number of iterations that are allowed for LP resolve</td>
<td>1000</td>
</tr>
<tr>
<td>lp/resolverestore</td>
<td>Should the LP be resolved to restore the state at start of diving (if FALSE we buffer the solution values)?</td>
<td>0</td>
</tr>
<tr>
<td>lp/rootiterlim</td>
<td>Iteration limit for initial root LP solve (-1: no limit)</td>
<td>-1</td>
</tr>
</tbody>
</table>
### 3.117 memory

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory/arraygrowfac</td>
<td>memory growing factor for dynamically allocated arrays</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 10]</td>
<td></td>
</tr>
<tr>
<td>memory/arraygrowinit</td>
<td>initial size of dynamically allocated arrays</td>
<td>4</td>
</tr>
<tr>
<td>memory/pathgrowfac</td>
<td>memory growing factor for path array</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 10]</td>
<td></td>
</tr>
<tr>
<td>memory/pathgrowinit</td>
<td>initial size of path array</td>
<td>256</td>
</tr>
<tr>
<td>memory/savefac</td>
<td>fraction of maximal memory usage resulting in switch to memory saving mode</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>memory/treegrowfac</td>
<td>memory growing factor for tree array</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Range: [1, 10]</td>
<td></td>
</tr>
<tr>
<td>memory/treegrowinit</td>
<td>initial size of tree array</td>
<td>65536</td>
</tr>
</tbody>
</table>

### 3.118 misc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>misc/calcintegral</td>
<td>should SCIP calculate the primal dual integral value?</td>
<td>1</td>
</tr>
<tr>
<td>misc/catchcrlc</td>
<td>should the CTRL-C interrupt be caught by SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>misc/estimexternmem</td>
<td>should the usage of external memory be estimated?</td>
<td>1</td>
</tr>
<tr>
<td>misc/finitesolutionstore</td>
<td>should SCIP try to remove infinite fixings from solutions copied to the solution store?</td>
<td>0</td>
</tr>
<tr>
<td>misc/improvingssols</td>
<td>should only solutions be checked which improve the primal bound</td>
<td>0</td>
</tr>
<tr>
<td>misc/permuteconss</td>
<td>seed value for permuting the problem after the problem was transformed (-1: no permutation)</td>
<td>-1</td>
</tr>
<tr>
<td>misc/permutevars</td>
<td>should order of constraints be permuted (depends on permutationseed)?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>should order of variables be permuted (depends on permutationseed)?</td>
<td>0</td>
</tr>
</tbody>
</table>
### misc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>misc/printreason</td>
<td>should the reason be printed if a given start solution is infeasible</td>
<td>0</td>
</tr>
<tr>
<td>misc/resetstat</td>
<td>should the statistics be reset if the transformed problem is freed (in case of a Benders decomposition this parameter should be set to FALSE)</td>
<td>1</td>
</tr>
<tr>
<td>misc/transorigsols</td>
<td>should SCIP try to transfer original solutions to the extended space (after presolving)?</td>
<td>1</td>
</tr>
<tr>
<td>misc/useconstable</td>
<td>should a hashtable be used to map from constraint names to constraints?</td>
<td>1</td>
</tr>
<tr>
<td>misc/usesmalltables</td>
<td>should smaller hashtables be used? yields better performance for small problems with about 100 variables</td>
<td>0</td>
</tr>
<tr>
<td>misc/usevartable</td>
<td>should a hashtable be used to map from variable names to variables?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.119 nodeselection

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/childsel</td>
<td>child selection rule ('d'own, 'u'p, 'p'seudo costs, 'i'nference, 'l'p value, 'r'oot LP value difference, 'h'ybrid inference/root LP value difference)</td>
<td>h</td>
</tr>
</tbody>
</table>

### 3.120 nodeselection/bfs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/bfs/maxplungedep</td>
<td>maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td>nodeselection/bfs/maxplungequot</td>
<td>maximal quotient (crlowerbound - lowerbound)/(cutoffbound - lowerbound) where plunging is performed</td>
<td>0.25</td>
</tr>
<tr>
<td>nodeselection/bfs/memsavepriority</td>
<td>priority of node selection rule &lt;bfs&gt; in memory saving mode Range: [-536870912, 536870911]</td>
<td>0</td>
</tr>
<tr>
<td>nodeselection/bfs/minplungedep</td>
<td>minimal plunging depth, before new best node may be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td>nodeselection/bfs/stdpriority</td>
<td>priority of node selection rule &lt;bfs&gt; in standard mode Range: [-536870912, 536870911]</td>
<td>100000</td>
</tr>
</tbody>
</table>

### 3.121 nodeselection/breadthfirst

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/breadthfirst/memsavepriority</td>
<td>priority of node selection rule &lt;breadthfirst&gt; in memory saving mode Range: [-536870912, 536870911]</td>
<td>-1000000</td>
</tr>
<tr>
<td>nodeselection/breadthfirst/stdpriority</td>
<td>priority of node selection rule &lt;breadthfirst&gt; in standard mode Range: [-536870912, 536870911]</td>
<td>-10000</td>
</tr>
</tbody>
</table>

### 3.122 nodeselection/dfs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/dfs/memsavepriority</td>
<td>priority of node selection rule &lt;dfs&gt; in memory saving mode Range: [-536870912, 536870911]</td>
<td>100000</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>nodeselection/dfs/stdpriority</td>
<td>priority of node selection rule <code>&lt;dfs&gt;</code> in standard mode</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: ([-536870912, 536870911])</td>
<td></td>
</tr>
</tbody>
</table>

### 3.123 nodeselection/estimate

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/estimate/bestnodefreq</td>
<td>frequency at which the best node instead of the best estimate is selected (0: never)</td>
<td>10</td>
</tr>
<tr>
<td>nodeselection/estimate/breadthfirstdepth</td>
<td>depth until breadth-first search is applied</td>
<td>-1</td>
</tr>
<tr>
<td>nodeselection/estimate/maxplungedepth</td>
<td>maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td>nodeselection/estimate/maxplungequot</td>
<td>maximal quotient (estimate - lowerbound)/(cutoffbound - lowerbound) where plunging is performed</td>
<td>0.25</td>
</tr>
<tr>
<td>nodeselection/estimate/memsavepriority</td>
<td>priority of node selection rule <code>&lt;estimate&gt;</code> in memory saving mode</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Range: ([-536870912, 536870911])</td>
<td></td>
</tr>
<tr>
<td>nodeselection/estimate/minplungedepth</td>
<td>minimal plunging depth, before new best node may be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td>nodeselection/estimate/plungeoffset</td>
<td>number of nodes before doing plunging the first time</td>
<td>0</td>
</tr>
<tr>
<td>nodeselection/estimate/stdpriority</td>
<td>priority of node selection rule <code>&lt;estimate&gt;</code> in standard mode</td>
<td>200000</td>
</tr>
<tr>
<td></td>
<td>Range: ([-536870912, 536870911])</td>
<td></td>
</tr>
</tbody>
</table>

### 3.124 nodeselection/hybridestim

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/hybridestim/bestnodefreq</td>
<td>frequency at which the best node instead of the hybrid best estimate / best bound is selected (0: never)</td>
<td>1000</td>
</tr>
<tr>
<td>nodeselection/hybridestim/estimweight</td>
<td>weight of estimate value in node selection score (0: pure best bound search, 1: pure best estimate search) Range: ([0, 1])</td>
<td>0.1</td>
</tr>
<tr>
<td>nodeselection/hybridestim/maxplungedepth</td>
<td>maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td>nodeselection/hybridestim/maxplungequot</td>
<td>maximal quotient (estimate - lowerbound)/(cutoffbound - lowerbound) where plunging is performed</td>
<td>0.25</td>
</tr>
<tr>
<td>nodeselection/hybridestim/memsavepriority</td>
<td>priority of node selection rule <code>&lt;hybridestim&gt;</code> in memory saving mode Range: ([-536870912, 536870911])</td>
<td>50</td>
</tr>
<tr>
<td>nodeselection/hybridestim/minplungedepth</td>
<td>minimal plunging depth, before new best node may be selected (-1 for dynamic setting)</td>
<td>-1</td>
</tr>
<tr>
<td>nodeselection/hybridestim/stdpriority</td>
<td>priority of node selection rule <code>&lt;hybridestim&gt;</code> in standard mode Range: ([-536870912, 536870911])</td>
<td>50000</td>
</tr>
</tbody>
</table>

### 3.125 nodeselection/restartdfs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/restartdfs/countonlyleaves</td>
<td>count only leaf nodes (otherwise all nodes)?</td>
<td>1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>nodeselection/restartdfs/memsavepriority</td>
<td>priority of node selection rule &lt;restartdfs&gt; in memory saving mode</td>
<td>50000</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
<tr>
<td>nodeselection/restartdfs/selectbestfreq</td>
<td>frequency for selecting the best node instead of the deepest one</td>
<td>100</td>
</tr>
<tr>
<td>nodeselection/restartdfs/stdpriority</td>
<td>priority of node selection rule &lt;restartdfs&gt; in standard mode</td>
<td>10000</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.126 nodeselection/uct

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeselection/uct/memsavepriority</td>
<td>priority of node selection rule &lt;uct&gt; in memory saving mode</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
<tr>
<td>nodeselection/uct/nodelimit</td>
<td>maximum number of nodes before switching to default rule</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1000000]</td>
<td></td>
</tr>
<tr>
<td>nodeselection/uct/stdpriority</td>
<td>priority of node selection rule &lt;uct&gt; in standard mode</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
<tr>
<td>nodeselection/uct/useestimate</td>
<td>should the estimate (TRUE) or lower bound of a node be used for UCT score?</td>
<td>0</td>
</tr>
<tr>
<td>nodeselection/uct/weight</td>
<td>weight for visit quotient of node selection rule</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.127 numerics

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>numerics/barrierconvtol</td>
<td>LP convergence tolerance used in barrier algorithm</td>
<td>1e-10</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-17, 0.001]</td>
<td></td>
</tr>
<tr>
<td>numerics/boundstreps</td>
<td>minimal relative improve for strengthening bounds</td>
<td>0.05</td>
</tr>
<tr>
<td>numerics/dualfeastol</td>
<td>feasibility tolerance for reduced costs in LP solution</td>
<td>1e-07</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-17, 0.001]</td>
<td></td>
</tr>
<tr>
<td>numerics/epsilon</td>
<td>absolute values smaller than this are considered zero</td>
<td>1e-09</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-20, 0.001]</td>
<td></td>
</tr>
<tr>
<td>numerics/feastol</td>
<td>feasibility tolerance for constraints</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-17, 0.001]</td>
<td></td>
</tr>
<tr>
<td>numerics/hugeval</td>
<td>values larger than this are considered huge and should be handled separately (e.g., in activity computation)</td>
<td>1e+15</td>
</tr>
<tr>
<td>numerics/lpfeastol</td>
<td>primal feasibility tolerance of LP solver</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-17, 0.001]</td>
<td></td>
</tr>
<tr>
<td>numerics/pseudocostdelta</td>
<td>minimal objective distance value to use for branching pseudo cost updates</td>
<td>0.0001</td>
</tr>
<tr>
<td>numerics/pseudocosteps</td>
<td>minimal variable distance value to use for branching pseudo cost updates</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-17, 1]</td>
<td></td>
</tr>
<tr>
<td>numerics/recomputefac</td>
<td>minimal decrease factor that causes the recomputation of a value (e.g., pseudo objective) instead of an update</td>
<td>1e+07</td>
</tr>
<tr>
<td>numerics/sumepsilon</td>
<td>absolute values of sums smaller than this are considered zero</td>
<td>1e-06</td>
</tr>
<tr>
<td></td>
<td>Range: [1e-17, 0.001]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.128 presolving
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/abortfac</td>
<td>abort presolve, if at most this fraction of the problem was changed in last</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>presolve round</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>presolving/donotaggr</td>
<td>should aggregation of variables be forbidden?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/donotmultaggr</td>
<td>should multi-aggregation of variables be forbidden?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/immrestartfac</td>
<td>fraction of integer variables that were fixed in the root node triggering</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>an immediate restart with preprocessing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>presolving/maxrestartfac</td>
<td>maximal number of restarts (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/maxrounds</td>
<td>maximal number of presolving rounds (-1: unlimited, 0: off)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/restartfac</td>
<td>fraction of integer variables that were fixed in the root node triggering a</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>restart with preprocessing after root node evaluation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>presolving/restartminred</td>
<td>minimal fraction of integer variables removed after restart to allow for an</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>additional restart</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>presolving/subrestartfac</td>
<td>fraction of integer variables that were globally fixed during the solving</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>process triggering a restart with preprocessing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.129 presolving/boundshift

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/boundshift/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/boundshift/flipping</td>
<td>is flipping allowed (multiplying with -1)?</td>
<td>1</td>
</tr>
<tr>
<td>presolving/boundshift/integer</td>
<td>shift only integer ranges?</td>
<td>1</td>
</tr>
<tr>
<td>presolving/boundshift/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>presolving/boundshift/maxshift</td>
<td>absolute value of maximum shift</td>
<td>maxint</td>
</tr>
<tr>
<td>presolving/boundshift/priority</td>
<td>priority of presolver &lt;boundshift&gt;</td>
<td>7900000</td>
</tr>
<tr>
<td></td>
<td>Range: [-536870912, 536870911]</td>
<td></td>
</tr>
</tbody>
</table>

### 3.130 presolving/components

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/components/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>presolving/components/feastolfactor</td>
<td>factor to increase the feasibility tolerance of the main SCIP in all sub-SCIPs, default value 1.0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1e+06]</td>
<td></td>
</tr>
<tr>
<td>presolving/components/intfactor</td>
<td>the weight of an integer variable compared to binary variables</td>
<td>1</td>
</tr>
<tr>
<td>presolving/components/maxintvars</td>
<td>maximum number of integer (or binary) variables to solve a subproblem directly (-1: unlimited)</td>
<td>500</td>
</tr>
<tr>
<td>presolving/components/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/components/nodelimit</td>
<td>maximum number of nodes to be solved in subproblems</td>
<td>10000</td>
</tr>
</tbody>
</table>
### 3.131 presolving/convertinttobin

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/convertinttobin/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/convertinttobin/maxdomainsize</td>
<td>absolute value of maximum domain size for converting an integer variable to binaries variables</td>
<td>maxint</td>
</tr>
<tr>
<td>presolving/convertinttobin/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>0</td>
</tr>
<tr>
<td>presolving/convertinttobin/onlypoweroftwo</td>
<td>should only integer variables with a domain size of $2^p - 1$ be converted, there we don’t need an knapsack-constraint for restricting the sum of the binaries</td>
<td>0</td>
</tr>
<tr>
<td>presolving/convertinttobin/priority</td>
<td>priority of presolver &lt;convertinttobin&gt;</td>
<td>6000000</td>
</tr>
<tr>
<td>presolving/convertinttobin/samemocksinbothdirections</td>
<td>should only integer variables with uplocks equals downlocks be converted</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.132 presolving/domcol

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/domcol/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>presolving/domcol/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/domcol/nummaxpairs</td>
<td>maximal number of pair comparisons Range: [1024, 1000000000]</td>
<td>1048576</td>
</tr>
<tr>
<td>presolving/domcol/numminpairs</td>
<td>minimal number of pair comparisons Range: [100, 1048576]</td>
<td>1024</td>
</tr>
<tr>
<td>presolving/domcol/predbndstr</td>
<td>should predictive bound strengthening be applied?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/domcol/priority</td>
<td>priority of presolver &lt;domcol&gt;</td>
<td>20000000</td>
</tr>
<tr>
<td>presolving/domcol/singcolstuffing</td>
<td>should singleton columns stuffing be applied?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.133 presolving/dualinfer

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/dualinfer/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>presolving/dualinfer/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>0</td>
</tr>
</tbody>
</table>
### 3.134 presolving/gateextraction

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/gateextraction/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>presolving/gateextraction/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/gateextraction/onlysetpart</td>
<td>should we only try to extract set-partitioning constraints and no and-constraints</td>
<td>0</td>
</tr>
<tr>
<td>presolving/gateextraction/priority</td>
<td>priority of presolver &lt;gateextraction&gt;</td>
<td>1000000</td>
</tr>
<tr>
<td>presolving/gateextraction/searchequations</td>
<td>should we try to extract set-partitioning constraint out of one logicor and one corresponding set-packing constraint</td>
<td>1</td>
</tr>
<tr>
<td>presolving/gateextraction/sorting</td>
<td>order logicor contraints to extract big-gates before smaller ones (-1), do not order them (0) or order them to extract smaller gates at first (1)</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.135 presolving/implics

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/implics/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/implics/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/implics/priority</td>
<td>priority of presolver &lt;implics&gt;</td>
<td>-10000</td>
</tr>
</tbody>
</table>

### 3.136 presolving/inttobinary

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/inttobinary/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/inttobinary/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>presolving/inttobinary/priority</td>
<td>priority of presolver &lt;inttobinary&gt;</td>
<td>7000000</td>
</tr>
</tbody>
</table>

### 3.137 presolving/trivial

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>presolving/trivial/delay</td>
<td>should presolver be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>presolving/trivial/maxrounds</td>
<td>maximal number of presolving rounds the presolver participates in (-1: no limit)</td>
<td>-1</td>
</tr>
</tbody>
</table>
### 3.138 propagating

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/abortoncutoff</td>
<td>should propagation be aborted immediately? setting this to FALSE could help conflict analysis to produce more conflict constraints</td>
<td>1</td>
</tr>
<tr>
<td>propagating/maxrounds</td>
<td>maximal number of propagation rounds per node (-1: unlimited)</td>
<td>100</td>
</tr>
<tr>
<td>propagating/maxroundsroot</td>
<td>maximal number of propagation rounds in the root node (-1: unlimited)</td>
<td>1000</td>
</tr>
</tbody>
</table>

### 3.139 propagating/dualfix

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/dualfix/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/dualfix/freq</td>
<td>frequency for calling propagator &lt;dualfix&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>propagating/dualfix/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/dualfix/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/dualfix/presolpriority</td>
<td>presolving priority of propagator &lt;dualfix&gt;</td>
<td>8000000</td>
</tr>
<tr>
<td>propagating/dualfix/priority</td>
<td>priority of propagator &lt;dualfix&gt;</td>
<td>8000000</td>
</tr>
<tr>
<td>propagating/dualfix/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: [1, 15]</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.140 propagating/genvbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/genvbounds/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/genvbounds/freq</td>
<td>frequency for calling propagator &lt;genvbounds&gt; (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/genvbounds/global</td>
<td>apply global propagation?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/genvbounds/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/genvbounds/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/genvbounds/presolpriority</td>
<td>presolving priority of propagator &lt;genvbounds&gt;</td>
<td>-2000000</td>
</tr>
<tr>
<td>propagating/genvbounds/priority</td>
<td>priority of propagator &lt;genvbounds&gt;</td>
<td>3000000</td>
</tr>
<tr>
<td>propagating/genvbounds/propinrootnode</td>
<td>apply genvbounds in root node if no new incumbent was found?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/genvbounds/sort</td>
<td>sort genvbounds and wait for bound change events?</td>
<td>1</td>
</tr>
</tbody>
</table>
### 3.141 propagating/obbt

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/obbt/applyfilterrounds</td>
<td>try to filter bounds in so-called filter rounds by solving auxiliary LPs?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/obbt/conditionlimit</td>
<td>maximum condition limit used in LP solver (-1.0: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/obbt/creategenvbounds</td>
<td>should obbt try to provide genvbounds if possible?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/obbt/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/obbt/dualfeastol</td>
<td>feasibility tolerance for reduced costs used in obbt; this value is used if SCIP’s dual feastol is greater</td>
<td>1e-09</td>
</tr>
<tr>
<td>propagating/obbt/freq</td>
<td>frequency for calling propagator &lt;obbt&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>propagating/obbt/itlimitfactor</td>
<td>multiple of root node LP iterations used as total LP iteration limit for obbt (≤ 0: no limit)</td>
<td>5</td>
</tr>
<tr>
<td>propagating/obbt/maxlookahead</td>
<td>maximal number of bounds evaluated without success per group (-1: no limit)</td>
<td>3</td>
</tr>
<tr>
<td>propagating/obbt/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/obbt/minfilter</td>
<td>minimal number of filtered bounds to apply another filter round</td>
<td>2</td>
</tr>
<tr>
<td>propagating/obbt/normalize</td>
<td>should coefficients in filtering be normalized w.r.t. the domains sizes?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/obbt/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/obbt/presolpriority</td>
<td>presolving priority of propagator &lt;obbt&gt;</td>
<td>0</td>
</tr>
<tr>
<td>propagating/obbt/priority</td>
<td>priority of propagator &lt;obbt&gt;</td>
<td>-1000000</td>
</tr>
<tr>
<td>propagating/obbt/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)</td>
<td>4</td>
</tr>
</tbody>
</table>

### 3.142 propagating/probing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/probing/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/probing/freq</td>
<td>frequency for calling propagator &lt;probing&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/probing/maxdepth</td>
<td>maximal depth until propagation is executed(-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/probing/maxfixings</td>
<td>maximal number of fixings found, until probing is interrupted (0: don’t interrupt)</td>
<td>25</td>
</tr>
<tr>
<td>propagating/probing/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/probing/maxruns</td>
<td>maximal number of runs, probing participates in (-1: no limit)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/probing/maxsumuseless</td>
<td>maximal number of probings without fixings, until probing is aborted (0: don’t abort)</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>propagating/probing/maxtotaluseless</td>
<td>maximal number of successive probings without fixings, bound changes, and implications, until probing is aborted (0: don’t abort)</td>
<td>50</td>
</tr>
<tr>
<td>propagating/probing/maxuseless</td>
<td>maximal number of successive probings without fixings, until probing is aborted (0: don’t abort)</td>
<td>1000</td>
</tr>
<tr>
<td>propagating/probing/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/probing/presolpriority</td>
<td>presolving priority of propagator &lt;probing&gt; &lt;probing&gt;</td>
<td>-100000</td>
</tr>
<tr>
<td>propagating/probing/priority</td>
<td>priority of propagator &lt;probing&gt; &lt;probing&gt;</td>
<td>-100000</td>
</tr>
<tr>
<td>propagating/probing/proprounds</td>
<td>maximal number of propagation rounds in probing subproblems (-1: no limit, 0: auto)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/probing/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS))</td>
<td>4</td>
</tr>
</tbody>
</table>

### 3.143 propagating/pseudoobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/pseudoobj/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/pseudoobj/force</td>
<td>should the propagator be forced even active pricer are present?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/pseudoobj/freq</td>
<td>frequency for calling propagator &lt;pseudoobj&gt; &lt;pseudoobj&gt;</td>
<td>1</td>
</tr>
<tr>
<td>propagating/pseudoobj/maximplvars</td>
<td>maximum number of binary variables the implications are used if turned on (-1: unlimited)?</td>
<td>50000</td>
</tr>
<tr>
<td>propagating/pseudoobj/maxnewvars</td>
<td>number of variable added after the propgator is reinitialized?</td>
<td>1000</td>
</tr>
<tr>
<td>propagating/pseudoobj/max prerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/pseudoobj/maxvarsfrac</td>
<td>maximal fraction of none binary variables with non-zero objective without a bound reduction before aborted</td>
<td>0.1</td>
</tr>
<tr>
<td>propagating/pseudoobj/minuseless</td>
<td>minimal number of successive none binary variable propagator without a bound reduction before aborted</td>
<td>100</td>
</tr>
<tr>
<td>propagating/pseudoobj/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/pseudoobj/presolpriority</td>
<td>presolving priority of propagator &lt;pseudoobj&gt; &lt;pseudoobj&gt;</td>
<td>6000000</td>
</tr>
<tr>
<td>propagating/pseudoobj/priority</td>
<td>priority of propagator &lt;pseudoobj&gt; &lt;pseudoobj&gt;</td>
<td>3000000</td>
</tr>
<tr>
<td>propagating/pseudoobj/propcutoffbound</td>
<td>propagate new cutoff bound directly globally</td>
<td>1</td>
</tr>
<tr>
<td>propagating/pseudoobj/propfullinroot</td>
<td>do we want to propagate all none binary variables if we are propagating the root node</td>
<td>1</td>
</tr>
<tr>
<td>propagating/pseudoobj/propuseimplics</td>
<td>use implications to strengthen the propagation of binary variable (increasing the objective change)?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/pseudoobj/respropuseimplics</td>
<td>use implications to strengthen the resolve propagation of binary variable (increasing the objective change)?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/pseudoobj/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS))</td>
<td>7</td>
</tr>
</tbody>
</table>
### 3.144 propagating/redcost

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/redcost/continuous</td>
<td>should reduced cost fixing be also applied to continuous variables?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/redcost/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/redcost/freq</td>
<td>frequency for calling propagator <code>&lt;redcost&gt;</code> (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/redcost/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/redcost/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/redcost/presolpriority</td>
<td>presolving priority of propagator <code>&lt;redcost&gt;</code> Range: [-536870912, 536870911]</td>
<td>0</td>
</tr>
<tr>
<td>propagating/redcost/priority</td>
<td>priority of propagator <code>&lt;redcost&gt;</code> Range: [-536870912, 536870911]</td>
<td>1000000</td>
</tr>
<tr>
<td>propagating/redcost/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>6</td>
</tr>
<tr>
<td>propagating/redcost/useimplics</td>
<td>should implications be used to strength the reduced cost for binary variables?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.145 propagating/rootredcost

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/rootredcost/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/rootredcost/freq</td>
<td>frequency for calling propagator <code>&lt;rootredcost&gt;</code> (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/rootredcost/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>propagating/rootredcost/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/rootredcost/presolpriority</td>
<td>presolving priority of propagator <code>&lt;rootredcost&gt;</code> Range: [-536870912, 536870911]</td>
<td>0</td>
</tr>
<tr>
<td>propagating/rootredcost/priority</td>
<td>priority of propagator <code>&lt;rootredcost&gt;</code> Range: [-536870912, 536870911]</td>
<td>1000000</td>
</tr>
<tr>
<td>propagating/rootredcost/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]</td>
<td>5</td>
</tr>
</tbody>
</table>

### 3.146 propagating/vbounds

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>propagating/vbounds/delay</td>
<td>should propagator be delayed, if other propagators found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/dotoposort</td>
<td>should the bounds be topologically sorted in advance?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/vbounds/freq</td>
<td>frequency for calling propagator <code>&lt;vbounds&gt;</code> (-1: never, 0: only in root node)</td>
<td>1</td>
</tr>
<tr>
<td>propagating/vbounds/maxprerounds</td>
<td>maximal number of presolving rounds the propagator participates in (-1: no limit)</td>
<td>-1</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>propagating/vbounds/presoldelay</td>
<td>should presolving be delayed, if other presolvers found reductions?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/presolpriority</td>
<td>presolving priority of propagator <code>&lt;vbounds&gt;</code>&lt;br&gt;Range: [-536870912, 536870911]</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/priority</td>
<td>priority of propagator <code>&lt;vbounds&gt;</code>&lt;br&gt;Range: [-536870912, 536870911]</td>
<td>3000000</td>
</tr>
<tr>
<td>propagating/vbounds/sortcliques</td>
<td>should cliques be regarded for the topological sort?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/timingmask</td>
<td>timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)&lt;br&gt;Range: [1, 15]</td>
<td>5</td>
</tr>
<tr>
<td>propagating/vbounds/usebdwidening</td>
<td>should bound widening be used to initialize conflict analysis?</td>
<td>1</td>
</tr>
<tr>
<td>propagating/vbounds/usecliques</td>
<td>should cliques be propagated?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/useimplics</td>
<td>should implications be propagated?</td>
<td>0</td>
</tr>
<tr>
<td>propagating/vbounds/usevbounds</td>
<td>should vbounds be propagated?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.147 separating

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/cutagelimit</td>
<td>maximum age a cut can reach before it is deleted from the global cut pool, or -1 to keep all cuts</td>
<td>100</td>
</tr>
<tr>
<td>separating/efficacynorm</td>
<td>row norm to use for efficacy calculation ('e'euclidean, 'm'maximum, 's'sum, 'd'discrete)</td>
<td>e</td>
</tr>
<tr>
<td>separating/feastolfac</td>
<td>factor on cut infeasibility to limit feasibility tolerance for relaxation solver (-1: off)&lt;br&gt;Range: [-1, 1]</td>
<td>-1</td>
</tr>
<tr>
<td>separating/maxaddrounds</td>
<td>maximal additional number of separation rounds in subsequent price-and-cut loops (-1: no additional restriction)</td>
<td>1</td>
</tr>
<tr>
<td>separating/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separation (0.0: only on current best node, 1.0: on all nodes)&lt;br&gt;Range: [0, 1]</td>
<td>1</td>
</tr>
<tr>
<td>separating/maxcuts</td>
<td>maximal number of cuts separated per separation round (0: disable local separation)</td>
<td>100</td>
</tr>
<tr>
<td>separating/maxcutroot</td>
<td>maximal number of separated cuts at the root node (0: disable root node separation)</td>
<td>2000</td>
</tr>
<tr>
<td>separating/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>separating/maxroundsroot</td>
<td>maximal number of separation rounds in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/maxroundsrootsubrun</td>
<td>maximal number of separation rounds in the root node of a subsequent run (-1: unlimited)</td>
<td>1</td>
</tr>
<tr>
<td>separating/maxruns</td>
<td>maximal number of runs for which separation is enabled (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/maxstallrounds</td>
<td>maximal number of consecutive separation rounds without objective or integrality improvement (-1: no additional restriction)</td>
<td>5</td>
</tr>
<tr>
<td>separating/minefficacy</td>
<td>minimal efficacy for a cut to enter the LP</td>
<td>0.05</td>
</tr>
<tr>
<td>separating/minefficacyroot</td>
<td>minimal efficacy for a cut to enter the LP in the root node</td>
<td>0.001</td>
</tr>
<tr>
<td>separating/minortho</td>
<td>minimal orthogonality for a cut to enter the LP&lt;br&gt;Range: [0, 1]</td>
<td>0.5</td>
</tr>
<tr>
<td>separating/minorthoroot</td>
<td>minimal orthogonality for a cut to enter the LP in the root node&lt;br&gt;Range: [0, 1]</td>
<td>0.5</td>
</tr>
<tr>
<td>separating/objparalfac</td>
<td>factor to scale objective parallelism of cut in separation score calculation</td>
<td>0.0001</td>
</tr>
</tbody>
</table>
### 3.148 separating/cgmip

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/cgmip/addviolationcons</td>
<td>add constraint to subscip that only allows violated cuts (otherwise add obj. limit)?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/addviolconshdlr</td>
<td>add constraint handler to filter out violated cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/allowlocal</td>
<td>Allow to generate local cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/cmirownbounds</td>
<td>tell CMIR-generator which bounds to used in rounding?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/conshdlrusenorm</td>
<td>should the violation constraint handler use the norm of a cut to check for feasibility?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/contconvert</td>
<td>Convert some integral variables to be continuous to reduce the size of the sub-MIP?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/contconvfrac</td>
<td>fraction of integral variables converted to be continuous (if contconvert) Range: ([0, 1])</td>
<td>0.1</td>
</tr>
<tr>
<td>separating/cgmip/contconvmin</td>
<td>minimum number of integral variables before some are converted to be continuous</td>
<td>100</td>
</tr>
<tr>
<td>separating/cgmip/cutcoefbnd</td>
<td>bounds on the values of the coefficients in the CG-cut</td>
<td>1000</td>
</tr>
<tr>
<td>separating/cgmip/decisiontree</td>
<td>Use decision tree to turn separation on/off?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/earlyterm</td>
<td>terminate separation if a violated (but possibly sub-optimal) cut has been found?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cgmip/freq</td>
<td>frequency for calling separator (&lt;\text{cgmip}&gt;) ((-1: \text{never}, 0: \text{only in root node}))</td>
<td>-1</td>
</tr>
<tr>
<td>separating/cgmip/intconvert</td>
<td>Convert some integral variables attaining fractional values to have integral value?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/intconvfrac</td>
<td>fraction of frac. integral variables converted to have integral value (if intconvert) Range: ([0, 1])</td>
<td>0.1</td>
</tr>
<tr>
<td>separating/cgmip/intconvmin</td>
<td>minimum number of integral variables before some are converted to have integral value</td>
<td>100</td>
</tr>
<tr>
<td>separating/cgmip/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator (&lt;\text{cgmip}&gt;) ((0.0: \text{only on current best node, 1.0: on all nodes})) Range: ([0, 1])</td>
<td>0</td>
</tr>
<tr>
<td>separating/cgmip/maxdepth</td>
<td>maximal depth at which the separator is applied ((-1: \text{unlimited}))</td>
<td>-1</td>
</tr>
<tr>
<td>separating/cgmip/maxnodelimit</td>
<td>maximum number of nodes considered for sub-MIP ((-1: \text{unlimited}))</td>
<td>5000</td>
</tr>
<tr>
<td>separating/cgmip/maxrounds</td>
<td>maximal number of cgmip separation rounds per node ((-1: \text{unlimited}))</td>
<td>5</td>
</tr>
</tbody>
</table>
### 3.149 separating/clique

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/clique/backtrackfreq</td>
<td>frequency for premature backtracking up to tree level 1 (0: no backtracking)</td>
<td>1000</td>
</tr>
<tr>
<td>separating/clique/cliquedensity</td>
<td>minimal density of cliques to use a dense clique table Range: [0, 1]</td>
<td>0.05</td>
</tr>
<tr>
<td>separating/clique/cliquetablemem</td>
<td>maximal memory size of dense clique table (in kb) Range: [0, 2.09715e+06]</td>
<td>20000</td>
</tr>
<tr>
<td>separating/clique/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/clique/freq</td>
<td>frequency for calling separator &lt;clique&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/clique/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;clique&gt; (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>separating/clique/maxsepacuts</td>
<td>maximal number of clique cuts separated per separation round (-1: no limit)</td>
<td>10</td>
</tr>
<tr>
<td>separating/clique/maxtreenodes</td>
<td>maximal number of nodes in branch and bound tree (-1: no limit)</td>
<td>10000</td>
</tr>
<tr>
<td>separating/clique/maxzeroextensions</td>
<td>maximal number of zero-valued variables extending the clique (-1: no limit)</td>
<td>1000</td>
</tr>
</tbody>
</table>
### 3.150 separating/closecuts

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/closecuts/closethres</td>
<td>threshold on number of generated cuts below which the ordinary separation is started</td>
<td>50</td>
</tr>
<tr>
<td>separating/closecuts/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/freq</td>
<td>frequency for calling separator &lt;closecuts&gt; (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/closecuts/inclobjcutoff</td>
<td>include an objective cutoff when computing the relative interior?</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;closecuts&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>1</td>
</tr>
<tr>
<td>separating/closecuts/maxlpiterfactor</td>
<td>factor for maximal LP iterations in relative interior computation compared to node LP iterations (negative for no limit)</td>
<td>10</td>
</tr>
<tr>
<td>separating/closecuts/maxunsuccessful</td>
<td>turn off separation in current node after unsuccessful calls (-1 never turn off)</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/priority</td>
<td>priority of separator &lt;closecuts&gt;</td>
<td>1000000</td>
</tr>
<tr>
<td>separating/closecuts/recompute</td>
<td>recompute relative interior point in each separation call?</td>
<td>0</td>
</tr>
<tr>
<td>separating/closecuts/seponomial</td>
<td>convex combination value for close cuts</td>
<td>0.3</td>
</tr>
<tr>
<td>separating/closecuts/separelint</td>
<td>generate close cuts w.r.t. relative interior point (best solution otherwise)?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.151 separating/cmir

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/cmir/aggrtol</td>
<td>tolerance for bound distances used to select continuous variable in current aggregated constraint to be eliminated</td>
<td>0.1</td>
</tr>
<tr>
<td>separating/cmir/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/cmir/densityoffset</td>
<td>additional number of variables allowed in row on top of density</td>
<td>100</td>
</tr>
<tr>
<td>separating/cmir/densityscore</td>
<td>weight of row density in the aggregation scoring of the rows</td>
<td>0.0001</td>
</tr>
<tr>
<td>separating/cmir/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cmir/fixedintegralrhs</td>
<td>should an additional variable be complemented if f0 = 0?</td>
<td>1</td>
</tr>
<tr>
<td>separating/cmir/freq</td>
<td>frequency for calling separator &lt;cmir&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/cmir/maxaggdensity</td>
<td>maximal density of aggregated row</td>
<td>0.2</td>
</tr>
<tr>
<td>separating/cmir/maxaggrs</td>
<td>maximal number of aggregations for each row per separation round</td>
<td>3</td>
</tr>
<tr>
<td>separating/cmir/fixaggrsroot</td>
<td>maximal number of aggregations for each row per separation round in the root node</td>
<td>6</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/cmir/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;cmir&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td>separating/cmir/maxconts</td>
<td>maximal number of active continuous variables in aggregated row</td>
<td>10</td>
</tr>
<tr>
<td>separating/cmir/maxcontsroot</td>
<td>maximal number of active continuous variables in aggregated row in the root node</td>
<td>10</td>
</tr>
<tr>
<td>separating/cmir/maxfails</td>
<td>maximal number of consecutive unsuccessful aggregation tries (-1: unlimited)</td>
<td>20</td>
</tr>
<tr>
<td>separating/cmir/maxfailsroot</td>
<td>maximal number of consecutive unsuccessful aggregation tries in the root node (-1: unlimited)</td>
<td>100</td>
</tr>
<tr>
<td>separating/cmir/maxrounds</td>
<td>maximal number of cmir separation rounds per node (-1: unlimited)</td>
<td>3</td>
</tr>
<tr>
<td>separating/cmir/maxroundsroot</td>
<td>maximal number of cmir separation rounds in the root node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td>separating/cmir/maxrowdensity</td>
<td>maximal density of row to be used in aggregation</td>
<td>0.05</td>
</tr>
<tr>
<td>separating/cmir/maxrowfac</td>
<td>maximal row aggregation factor</td>
<td>10000</td>
</tr>
<tr>
<td>separating/cmir/maxsepacuts</td>
<td>maximal number of cmir cuts separated per separation round</td>
<td>100</td>
</tr>
<tr>
<td>separating/cmir/maxsepacutsroot</td>
<td>maximal number of cmir cuts separated per separation round in the root node</td>
<td>500</td>
</tr>
<tr>
<td>separating/cmir/maxslack</td>
<td>maximal slack of rows to be used in aggregation</td>
<td>0</td>
</tr>
<tr>
<td>separating/cmir/maxslackroot</td>
<td>maximal slack of rows to be used in aggregation in the root node</td>
<td>0.1</td>
</tr>
<tr>
<td>separating/cmir/maxtestdelta</td>
<td>maximal number of different deltas to try (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/cmir/maxtries</td>
<td>maximal number of rows to start aggregation with per separation round (-1: unlimited)</td>
<td>100</td>
</tr>
<tr>
<td>separating/cmir/maxtriesroot</td>
<td>maximal number of rows to start aggregation with per separation round in the root node (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/cmir/priority</td>
<td>priority of separator &lt;cmir&gt;</td>
<td>-3000</td>
</tr>
<tr>
<td>separating/cmir/slackscore</td>
<td>weight of slack in the aggregation scoring of the rows</td>
<td>0.001</td>
</tr>
<tr>
<td>separating/cmir/trynegscaling</td>
<td>should negative values also be tested in scaling?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.152 separating/flowcover

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/flowcover/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/flowcover/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/flowcover/freq</td>
<td>frequency for calling separator &lt;flowcover&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/flowcover/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;flowcover&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td>separating/flowcover/maxfails</td>
<td>maximal number of consecutive fails to generate a cut per separation round (-1: unlimited)</td>
<td>50</td>
</tr>
<tr>
<td>separating/flowcover/maxfailsroot</td>
<td>maximal number of consecutive fails to generate a cut per separation round in the root (-1: unlimited)</td>
<td>100</td>
</tr>
</tbody>
</table>
### separating/flowcover

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/flowcover/maxrounds</td>
<td>maximal number of separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>separating/flowcover/maxroundsroot</td>
<td>maximal number of separation rounds in the root node (-1: unlimited)</td>
<td>15</td>
</tr>
<tr>
<td>separating/flowcover/maxrowdensity</td>
<td>maximal density of row to separate flow cover cuts for</td>
<td>1</td>
</tr>
<tr>
<td>separating/flowcover/maxsepacuts</td>
<td>maximal number of flow cover cuts separated per separation round</td>
<td>100</td>
</tr>
<tr>
<td>separating/flowcover/maxsepacutsroot</td>
<td>maximal number of flow cover cuts separated per separation round in the root</td>
<td>200</td>
</tr>
<tr>
<td>separating/flowcover/maxslack</td>
<td>maximal slack of rows to separate flow cover cuts for</td>
<td>maxdouble</td>
</tr>
<tr>
<td>separating/flowcover/maxslackroot</td>
<td>maximal slack of rows to separate flow cover cuts for in the root</td>
<td>maxdouble</td>
</tr>
<tr>
<td>separating/flowcover/maxtestdelta</td>
<td>cut generation heuristic: maximal number of different deltas to try</td>
<td>10</td>
</tr>
<tr>
<td>separating/flowcover/maxtries</td>
<td>maximal number of rows to separate flow cover cuts for per separation round (-1: unlimited)</td>
<td>100</td>
</tr>
<tr>
<td>separating/flowcover/maxtriesroot</td>
<td>maximal number of rows to separate flow cover cuts for per separation round in the root (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/flowcover/multbyminusone</td>
<td>should flow cover cuts be separated for 0-1 single node flow set with reversed arcs in addition?</td>
<td>1</td>
</tr>
<tr>
<td>separating/flowcover/priority</td>
<td>priority of separator &lt;flowcover&gt;</td>
<td>-4000</td>
</tr>
<tr>
<td>separating/flowcover/slackscore</td>
<td>weight of slack in the scoring of the rows</td>
<td>0.001</td>
</tr>
</tbody>
</table>

### separating/gomory

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/gomory/away</td>
<td>minimal integrality violation of a basis variable in order to try Gomory cut</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Range: [0.0001, 0.5]</td>
<td></td>
</tr>
<tr>
<td>separating/gomory/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/gomory/delayedcuts</td>
<td>should cuts be added to the delayed cut pool?</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/forcecuts</td>
<td>if conversion to integral coefficients failed still consider the cut</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/freq</td>
<td>frequency for calling separator &lt;gomory&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/gomory/makeintegral</td>
<td>try to scale cuts to integral coefficients</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;gomory&gt; (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Range: [0, 1]</td>
<td></td>
</tr>
<tr>
<td>separating/gomory/maxrank</td>
<td>maximal rank of a gomory cut that could not be scaled to integral coefficients (-1: unlimited)</td>
<td>3</td>
</tr>
<tr>
<td>separating/gomory/maxrankintegral</td>
<td>maximal rank of a gomory cut that could be scaled to integral coefficients (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/gomory/maxrounds</td>
<td>maximal number of gomory separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>separating/gomory/maxroundsroot</td>
<td>maximal number of gomory separation rounds in the root node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/gomory/maxsepacuts</td>
<td>maximal number of gomory cuts separated per separation round</td>
<td>50</td>
</tr>
<tr>
<td>separating/gomory/maxsepacutscroot</td>
<td>maximal number of gomory cuts separated per separation round in the root node</td>
<td>200</td>
</tr>
<tr>
<td>separating/gomory/maxweightrange</td>
<td>maximal valid range \max(</td>
<td>\text{weights}</td>
</tr>
<tr>
<td>separating/gomory/priority</td>
<td>priority of separator \texttt{&lt;gomory&gt;}</td>
<td>-1000</td>
</tr>
<tr>
<td>separating/gomory/separaterows</td>
<td>separate rows with integral slack</td>
<td>1</td>
</tr>
<tr>
<td>separating/gomory/sidetypebasis</td>
<td>choose side types of row (lhs/rhs) based on basis information?</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.154 separating/impliedbounds

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/impliedbounds/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/impliedbounds/freq</td>
<td>frequency for calling separator \texttt{&lt;impliedbounds&gt;} (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/impliedbounds/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator \texttt{&lt;impliedbounds&gt;} (0.0: only on current best node, 1.0: on all nodes) RANGE: ([0, 1])</td>
<td>0</td>
</tr>
<tr>
<td>separating/impliedbounds/priority</td>
<td>priority of separator \texttt{&lt;impliedbounds&gt;}</td>
<td>-50</td>
</tr>
</tbody>
</table>

### 3.155 separating/intobj

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/intobj/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/intobj/freq</td>
<td>frequency for calling separator \texttt{&lt;intobj&gt;} (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/intobj/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator \texttt{&lt;intobj&gt;} (0.0: only on current best node, 1.0: on all nodes) RANGE: ([0, 1])</td>
<td>0</td>
</tr>
<tr>
<td>separating/intobj/priority</td>
<td>priority of separator \texttt{&lt;intobj&gt;}</td>
<td>-100</td>
</tr>
</tbody>
</table>

### 3.156 separating/mcf

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/mcf/checkcutshoredisconnectivity</td>
<td>should we separate only if the cuts shores are connected?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/mcf/dynamiccarts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/fixintegrallrhss</td>
<td>should an additional variable be complemented if f0 = 0?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/freq</td>
<td>frequency for calling separator \texttt{&lt;mcf&gt;} (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/mcf/maxarcinconsistencyratio</td>
<td>maximum inconsistency ratio of arcs not to be deleted</td>
<td>0.5</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>separating/mcf/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator (&lt;mcf&gt;) (0.0: only on current best node, 1.0: on all nodes) (\text{Range: } [0, 1])</td>
<td>0</td>
</tr>
<tr>
<td>separating/mcf/maxinconsistencyratio</td>
<td>maximum inconsistency ratio for separation at all</td>
<td>0.02</td>
</tr>
<tr>
<td>separating/mcf/maxsepacuts</td>
<td>maximal number of mcf cuts separated per separation round</td>
<td>100</td>
</tr>
<tr>
<td>separating/mcf/maxsepacutsroot</td>
<td>maximal number of mcf cuts separated per separation round in the root node – default separation</td>
<td>200</td>
</tr>
<tr>
<td>separating/mcf/maxtestdelta</td>
<td>maximal number of different deltas to try (-1: unlimited) – default separation</td>
<td>20</td>
</tr>
<tr>
<td>separating/mcf/maxweightrange</td>
<td>maximal valid range (\max(</td>
<td>\text{weights}</td>
</tr>
<tr>
<td>separating/mcf/modeltype</td>
<td>model type of network (0: auto, 1: directed, 2: undirected) (\text{Range: } [0, 2])</td>
<td>0</td>
</tr>
<tr>
<td>separating/mcf/nclusters</td>
<td>number of clusters to generate in the shrunken network – default separation (\text{Range: } [2, 32])</td>
<td>5</td>
</tr>
<tr>
<td>separating/mcf/priority</td>
<td>priority of separator (&lt;mcf&gt;) (\text{Range: } [-536870912, 536870911])</td>
<td>-10000</td>
</tr>
<tr>
<td>separating/mcf/separateflowcutset</td>
<td>should we separate flowcutset inequalities on the network cuts?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/separateknapsack</td>
<td>should we separate knapsack cover inequalities on the network cuts?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/separatesinglenodecuts</td>
<td>should we separate inequalities based on single-node cuts?</td>
<td>1</td>
</tr>
<tr>
<td>separating/mcf/trynegscaling</td>
<td>should negative values also be tested in scaling?</td>
<td>0</td>
</tr>
</tbody>
</table>

3.157 separating/oddcycle

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/oddcycle/addselfarcs</td>
<td>add links between a variable and its negated</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/allowmultiplecuts</td>
<td>even if a variable is already covered by a cut, still allow another cut to cover it too</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/cutthreshold</td>
<td>maximal number of other cuts s.t. separation is applied (-1 for direct call)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/oddcycle/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/freq</td>
<td>frequency for calling separator (&lt;\text{oddcycle}&gt;) (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/oddcycle/includetriangles</td>
<td>separate triangles found as 3-cycles or repaired larger cycles</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/liftdodcycles</td>
<td>should odd cycle cuts be lifted?</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/lpliftcoef</td>
<td>choose lifting candidate by coef*lpvalue or only by coef</td>
<td>0</td>
</tr>
<tr>
<td>separating/oddcycle/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator (&lt;\text{oddcycle}&gt;) (0.0: only on current best node, 1.0: on all nodes) (\text{Range: } [0, 1])</td>
<td>1</td>
</tr>
<tr>
<td>separating/oddcycle/maxcutslevel</td>
<td>maximal number of oddcycle cuts generated in every level of the level graph</td>
<td>50</td>
</tr>
<tr>
<td>separating/oddcycle/maxcutsroot</td>
<td>maximal number of oddcycle cuts generated per chosen variable as root of the level graph</td>
<td>1</td>
</tr>
</tbody>
</table>
### separating/oddcycle

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxnlevels</td>
<td>maximal number of levels in level graph</td>
<td>20</td>
</tr>
<tr>
<td>maxpernodeslevel</td>
<td>percentage of nodes allowed in the same level of the level graph [0-100]</td>
<td>100</td>
</tr>
<tr>
<td>maxreference</td>
<td>minimal weight on an edge (in level graph or bipartite graph)</td>
<td>0</td>
</tr>
<tr>
<td>maxrounds</td>
<td>maximal number of oddcycle separation rounds per node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td>maxroundsroot</td>
<td>maximal number of oddcycle separation rounds in the root node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td>maxsepacuts</td>
<td>maximal number of oddcycle cuts separated per separation round</td>
<td>5000</td>
</tr>
<tr>
<td>maxsepacutsroot</td>
<td>maximal number of oddcycle cuts separated per separation round in the root node</td>
<td>5000</td>
</tr>
<tr>
<td>maxunsuccesfull</td>
<td>number of unsuccessful calls at current node</td>
<td>3</td>
</tr>
<tr>
<td>multiplecuts</td>
<td>even if a variable is already covered by a cut, still try it as start node for a cycle search</td>
<td>0</td>
</tr>
<tr>
<td>offsetnodeslevel</td>
<td>offset of nodes allowed in the same level of the level graph (additional to the percentage of levelnodes)</td>
<td>10</td>
</tr>
<tr>
<td>offsettestvars</td>
<td>offset of variables to try the chosen method on (additional to the percentage of testvars)</td>
<td>100</td>
</tr>
<tr>
<td>percenttestvars</td>
<td>percentage of variables to try the chosen method on [0-100]</td>
<td>0</td>
</tr>
<tr>
<td>priority</td>
<td>priority of separator &lt;oddcycle&gt;</td>
<td>-15000</td>
</tr>
<tr>
<td>recalcliftcoef</td>
<td>calculate lifting coefficient of every candidate in every step (or only if its chosen)</td>
<td>1</td>
</tr>
<tr>
<td>repaircycles</td>
<td>try to repair violated cycles with double appearance of a variable</td>
<td>1</td>
</tr>
<tr>
<td>scalingfactor</td>
<td>factor for scaling of the arc-weights</td>
<td>1000</td>
</tr>
<tr>
<td>sortrootneighbors</td>
<td>sort level of the root neighbors by fractionality (maxfrac)</td>
<td>1</td>
</tr>
<tr>
<td>sortswitch</td>
<td>use sorted variable array (unsorted(0),maxlp(1),minlp(2),maxfrac(3),minfrac(4))</td>
<td>3</td>
</tr>
<tr>
<td>usegls</td>
<td>should the search method by Groetschel, Lovasz, Schrijver be used? Otherwise use levelgraph method by Hoffman, Padberg.</td>
<td>1</td>
</tr>
</tbody>
</table>

### separating/rapidlearning

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>applybdchgs</td>
<td>should the found global bound deductions be applied in the original SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>applyconflicts</td>
<td>should the found conflicts be applied in the original SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>applyinervals</td>
<td>should the inference values be used as initialization in the original SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>applyprimalsol</td>
<td>should the incumbent solution be copied to the original SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>applysolved</td>
<td>should a solved status be copied to the original SCIP?</td>
<td>1</td>
</tr>
<tr>
<td>contvars</td>
<td>should rapid learning be applied when there are continuous variables?</td>
<td>0</td>
</tr>
<tr>
<td>contvarsquot</td>
<td>maximal portion of continuous variables to apply rapid learning</td>
<td>0.3</td>
</tr>
</tbody>
</table>
separating/rapidlearning/copycuts should all active cuts from cutpool be copied to constraints in subproblem? 1
separating/rapidlearning/delay should separator be delayed, if other separators found cuts? 0
separating/rapidlearning/freq frequency for calling separator <rapidlearning> (-1: never, 0: only in root node) -1
separating/rapidlearning/lpiterquot maximal fraction of LP iterations compared to node LP iterations 0.2
separating/rapidlearning/maxbounddist maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator <rapidlearning> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1] 1
separating/rapidlearning/maxnconss maximum problem size (constraints) for which rapid learning will be called 10000
separating/rapidlearning/maxnodes maximum number of nodes considered in rapid learning run 5000
separating/rapidlearning/maxnvars maximum problem size (variables) for which rapid learning will be called 10000
separating/rapidlearning/minnodes minimum number of nodes considered in rapid learning run 500
separating/rapidlearning/priority priority of separator <rapidlearning> Range: [−536870912, 536870911] -1200000
separating/rapidlearning/reducedinfer should the inference values only be used when rapidlearning found other reductions? 0

3.159 separating/strongcg

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/strongcg/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/strongcg/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/strongcg/freq</td>
<td>frequency for calling separator &lt;strongcg&gt; (-1: never, 0: only in root node)</td>
<td>0</td>
</tr>
<tr>
<td>separating/strongcg/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator &lt;strongcg&gt; (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]</td>
<td>0</td>
</tr>
<tr>
<td>separating/strongcg/maxrounds</td>
<td>maximal number of strong CG separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>separating/strongcg/maxroundsroot</td>
<td>maximal number of strong CG separation rounds in the root node (-1: unlimited)</td>
<td>20</td>
</tr>
<tr>
<td>separating/strongcg/maxsepacuts</td>
<td>maximal number of strong CG cuts separated per separation round</td>
<td>50</td>
</tr>
<tr>
<td>separating/strongcg/maxsepacutsroot</td>
<td>maximal number of strong CG cuts separated per separation round in the root node</td>
<td>500</td>
</tr>
<tr>
<td>separating/strongcg/maxweightrange</td>
<td>maximal valid range max(</td>
<td>weights</td>
</tr>
<tr>
<td>separating/strongcg/priority</td>
<td>priority of separator &lt;strongcg&gt; Range: [−536870912, 536870911]</td>
<td>-2000</td>
</tr>
</tbody>
</table>

3.160 separating/zerohalf
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/zerohalf/delay</td>
<td>should separator be delayed, if other separators found cuts?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/delayedcuts</td>
<td>should cuts be added to the delayed cut pool?</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/dynamiccuts</td>
<td>should generated cuts be removed from the LP if they are no longer tight?</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/freq</td>
<td>frequency for calling separator <code>&lt;zerohalf&gt;</code> (-1: never, 0: only in root node)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/zerohalf/ignoreprevzhcuts</td>
<td>should zerohalf cuts found in previous callbacks ignored?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/maxbounddist</td>
<td>maximal relative distance from current node’s dual bound to primal bound compared to best node’s dual bound for applying separator <code>&lt;zerohalf&gt;</code> (0.0: only on current best node, 1.0: on all nodes)</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/maxcutfound</td>
<td>maximal number of {0,1/2}-cuts determined per separation round # (this includes separated but inefficacious cuts)</td>
<td>100</td>
</tr>
<tr>
<td>separating/zerohalf/maxcutfoundroot</td>
<td>maximal number of {0,1/2}-cuts determined per separation round in the root node # (this includes separated but inefficacious cuts)</td>
<td>1000</td>
</tr>
<tr>
<td>separating/zerohalf/maxdepth</td>
<td>separating cuts only if depth \leq maxdepth (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/zerohalf/maxncalls</td>
<td>maximal number of calls (-1: unlimited)</td>
<td>-1</td>
</tr>
<tr>
<td>separating/zerohalf/maxrounds</td>
<td>maximal number of zerohalf separation rounds per node (-1: unlimited)</td>
<td>5</td>
</tr>
<tr>
<td>separating/zerohalf/maxroundsroot</td>
<td>maximal number of zerohalf separation rounds in the root node (-1: unlimited)</td>
<td>10</td>
</tr>
<tr>
<td>separating/zerohalf/maxsepacuts</td>
<td>maximal number of {0,1/2}-cuts separated per separation round</td>
<td>50</td>
</tr>
<tr>
<td>separating/zerohalf/maxsepacutsroot</td>
<td>maximal number of {0,1/2}-cuts separated per separation round in the root node</td>
<td>500</td>
</tr>
<tr>
<td>separating/zerohalf/maxtestdelta</td>
<td>maximal number of different deltas to try for cmir (-1: unlimited, 0: delta=1)</td>
<td>10</td>
</tr>
<tr>
<td>separating/zerohalf/onlyorigrows</td>
<td>should only original LP rows be considered (i.e. ignore previously added LP rows)?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/priority</td>
<td>priority of separator <code>&lt;zerohalf&gt;</code></td>
<td>-6000</td>
</tr>
<tr>
<td>separating/zerohalf/relaxcontvars</td>
<td>should continuous variables be relaxed by adding variable bounds?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/scalefraccoeffs</td>
<td>should rows be scaled to make fractional coefficients integer?</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/trynegscaling</td>
<td>should negative values also be tested in scaling for cmir?</td>
<td>1</td>
</tr>
<tr>
<td>separating/zerohalf/usezhcutpool</td>
<td>should zerohalf cuts be filtered using a cutpool?</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3.161 separating/zerohalf/preprocessing

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/zerohalf/preprocessing/decomposeproblem</td>
<td>should problem be decomposed into subproblems (if possible) before applying preprocessing?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/preprocessing/delta</td>
<td>value of delta parameter used in preprocessing method ‘d’</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Range: \([0, 1]\)
preprocessing methods and ordering:
# 'd' columns with small LP solution, # 'G' modified Gaussian elimination, # 'i' identical columns, # 'I' identical rows, # 'L' large slack rows, # 'M' large slack rows (minslack), # 's' column singletons, # 'X' add trivial zerohalf cuts, # 'z' zero columns, # 'Z' zero rows, # 'C' fast \{'z','s'\}, # 'R' fast \{'Z','L','I'\} # # '-' no preprocessing #

3.162 separating/zerohalf/separating

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/zerohalf/separating/forcecutstolp</td>
<td>should the cuts be forced to enter the LP?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/separating/forcecutstosepastore</td>
<td>should the cuts be forced to enter SCIP's sepastore?</td>
<td>0</td>
</tr>
<tr>
<td>separating/zerohalf/separating/minviolation</td>
<td>minimal violation of a {0,1/2}-cut to be separated Range: [0.001, 0.5]</td>
<td>0.3</td>
</tr>
<tr>
<td>separating/zerohalf/separating/sepamethods</td>
<td>separating methods and ordering: # '!' stop further processing if a cut was found, # '2' exact polynomial time algorithm (only if matrix has max 2 odd entries per row), # 'e' enumeration heuristics (k=1: try all preprocessed rows), # 'E' enumeration heuristics (k=2: try all combinations of up to two preprocessed rows), # 'g' Extended Gaussian elimination heuristics, # 's' auxiliary IP heuristics (i.e. number of solved nodes is limited) # 'S' auxiliary IP exact (i.e. unlimited number of nodes) # # '-' no processing #</td>
<td>2g</td>
</tr>
</tbody>
</table>

3.163 separating/zerohalf/separating/auxip

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>separating/zerohalf/separating/auxip/objective</td>
<td>auxiliary IP objective: # 'v' maximize cut violation, # 'u' minimize number of aggregated rows in cut, # 'w' minimize number of aggregated rows in cut # weighted by the number of rows in the aggregation, # 'p' maximize cut violation and penalize a high number # of aggregated rows in the cut weighted by the number # of rows in the aggregation and the penalty factor p #</td>
<td>v</td>
</tr>
<tr>
<td>separating/zerohalf/separating/auxip/penaltyfactor</td>
<td>penalty factor used with objective function 'p' of auxiliary IP Range: [0.001]</td>
<td>0.001</td>
</tr>
<tr>
<td>separating/zerohalf/separating/auxip/settingsfile</td>
<td>optional settings file of the auxiliary IP (-: none)</td>
<td>-</td>
</tr>
<tr>
<td>separating/zerohalf/separating/auxip/sollimit</td>
<td>limits/solutions setting of the auxiliary IP</td>
<td>-1</td>
</tr>
<tr>
<td>separating/zerohalf/separating/auxip/useallsols</td>
<td>should all (proper) solutions of the auxiliary IP be used to generate cuts instead of using only the best?</td>
<td>1</td>
</tr>
</tbody>
</table>

3.164 timing
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>timing/clocktype</td>
<td>default clock type (1: CPU user seconds, 2: wall clock time) Range: [1, 2]</td>
<td>1</td>
</tr>
<tr>
<td>timing/enabled</td>
<td>is timing enabled?</td>
<td>1</td>
</tr>
<tr>
<td>timing/rareclockcheck</td>
<td>should clock checks of solving time be performed less frequently (note: time limit could be exceeded slightly)</td>
<td>0</td>
</tr>
<tr>
<td>timing/reading</td>
<td>belongs reading time to solving time?</td>
<td>0</td>
</tr>
<tr>
<td>timing/statistictiming</td>
<td>should timing for statistic output be performed?</td>
<td>1</td>
</tr>
</tbody>
</table>

### Bibliography


