GAMS — The Solver Manuals

Contents

Basic Solver Usage

Contents

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 that are 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 trnsport 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 follow. Included with each option is a description of how this option is interpreted by a GAMS solver.

Option	**Description**
`iterlim`	Sets a limit on the simplex iterations (i.e. pivots) performed by the solver. If this limit is hit, the solver will terminate and return solver status `2 ITERATION INTERRUPT`. Note that this option does not apply to other types of iterations (e.g. barrier iterations, major iterations in a nonlinear solver). These limits must be set by solver-specific options. In case many subproblems are solved via pivotal methods (e.g. in Branch and Bound or in an NLP solver), `iterlim` may be used as either a per-subproblem or cumulative pivot limit: this is solver dependent.
`reslim`	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.
`optfile`	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`*`.o67`, `optfile=525` implies *`solvername`*`.525`, etc.
`nodlim`	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`.
`optca`	MIP absolute optimality criterion. The absolute gap is defined to be $\|BP - BF\|$, where the best found value BF is the objective function value of the best integer solution found thus far and the best possible value BP is the current bound on the problem's solution. If the absolute gap is no greater than `optca`, the solver will terminate and return solver status `1 NORMAL COMPLETION` and model status `8 INTEGER SOLUTION`. Note that this is a termination test only; setting this option should not change the global search.
`optcr`	MIP relative optimality criterion. The relative gap is defined to be $\|BP - BF\|/\|BP\|$. If the relative gap is no greater than `optcr`, the solver will terminate and return solver status `1 NORMAL COMPLETION` and model status `8 INTEGER SOLUTION`. Note that this is a termination test only; setting this option should not change the global search. Note also that the relative gap is defined only if BP and BF have the same (nonzero) sign; if this is not the case, the `optcr` termination test will not be made.
`prioropt`	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.
`cheat`	MIP cheat value: Each new integer 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 minimizattion and maximization models. Using the `cheat` option invalidates any reporting of the best bound or optimality gaps.
`cutoff`	Cutoff value: When the branch and bound search starts, 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 integer solutions whose value is worse than `cutoff`.
`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.
`bratio`	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. All the pivotal algorithms in GAMS solvers will make use of this advanced basis to speed up problem solution.
`domlim`	Sets the domain violation limit. Domain errors are evaluation errors in the nonlinear functions (e.g. $\sqrt{x}$ for $x < 0$). When a domain violation occurs the domain error count is increased by one; 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.

Option	Description
`sysout`	If `option sysout=on` 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 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`)
`workfactor`	Specifies a factor to be applied to the solver-computed memory estimate. E.g. setting `workfactor=2` doubles the memory estimate. In case a solver allocates memory dynamically as it is needed, this option will have no affect. In case `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 case a solver allocates memory dynamically as it is needed, this option will have no affect. `workspace` exists only as a model-specific option.

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,

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

The option file takes its name from the solver being used: *solvername.XXX*, where '*solvername*' 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,

`optfile` model suffix value	Name of option file
0	No option file used
1	*solvername*.opt
2	*solvername*.op2
3	*solvername*.op3
10	*solvername*.o10
91	*solvername*.o91
100	*solvername*.100
999	*solvername*.999

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 file.

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 solver 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 key word `'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.

AlphaECP

Tapio Westerlund and Toni Lastusilta, Åbo Akademi University, Finland, twesterl@abo.fi

Contents

1 Introduction

GAMS/AlphaECP is a MINLP (Mixed-Integer Non-Linear Programming) solver based on the extended cutting plane (ECP) method. The solver can be applied to general MINLP problems and global optimal solutions can be ensured 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 (Kelley, 1960). The method requires only the solution of a MIP sub problem in each iteration. The MIP sub problems may be solved to optimality, but can also be solved 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 Westerlund T. and Pörn R. (2002). Solving Pseudo-Convex Mixed Integer Optimization Problems by Cutting Plane Techniques. *Optimization and Engngineering*, 3. 253-280.

The GAMS/AlphaECP algorithm has been further developed by introducing some additional functionality. A NLP solver can be called at MIP solutions which improve AlphaECP in finding feasible and accurate solutions, especially for MINLP problems, containing mainly of continuous variables. Furhermore, 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

In order to use GAMS/AlphaECP, users will need to have a GAMS/AlphaECP license. Additionally a licensed MIP solver is required for solving the mixed integer subproblems. If the NLP option is used a licensed NLP solver is additionally required.

1.2 Running GAMS/AlphaECP

GAMS/AlphaECP solves MINLP models. If GAMS/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, GAMS/AlphaECP can also handle NLP models, but is more suitable for MINLP problems. Especially in combination with an NLP solver it can find solutions the NLP solver by itself does not find. In this case it acts as a good starting point generator. If you want to solve NLPs with GAMS/AlphaECP you need to *trick* the GAMS system by solving your NLP as an MINLP:

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

Throughout the progress of GAMS/AlphaECP and at the end of the algorithm, constraint violations are reported. The violation reported is 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:

```
AlphaECP         ALFA 19Jun11 23.8.0 WIN 26392.26396 VS8 x86/MS Windows

------------------------------------------------------------------------------
                  Welcome to Alpha-ECP v2.04.02
 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
 Westerlund Tapio and Poern Ray (2002). Optimization & Engineering, 3, 253-280
-------------------------------------------------------------------------------
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)
composed 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
Solvelink for NLP and MIP subsolver                    solvelink=5
Cutting plane strategy (0-3)                           CUTdelcrit=3
Cut generation pace                                    CUTnrcuts=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=0
Level of AlphaECP output to statusfile (0-4)           ECPloglevel=0
User specified startpoint (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 subproblems (0->1.0)  MIPoptcr=1.00
Initial MIPoptcr interval before MIPoptcr reduction    MIPoptcrlim=200
```

```
Strategy for multiple MIP solutions                      MIPsolstrat=1
MIP solver for subproblems                               MIPsolver=cplex
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 subproblems                               NLPsolver=conopt
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
--------------------------------------------------------------------------------
Itera  Stepcode, Number  Point Alpha OPT Movement Viol  Maximum   MIPobjval
tion   Problems  of Cuts usage Upd.  CR  Norm     Cons  Violation
Startpoint: NL constraint (1) infeasibile
0      H         0       0     0     1   0        4     1.8E+003  NA
1      SAFGI     1       1     1     1   9.3E+003 0     0         8566.12
1      FOUND SOLUTION:   8566.12      (NLP) in 1 sec.

2      SAFH      1       1     0     1   6.6E+003 4     1.8E+003  4844.02
3      SAFH      3       2     0     1   9.3E+003 4     1.8E+003  4844.02
4      SAH       4       3     0     1   3.6E+003 4     9E+002    4844.02
5      SAFH      7       4     0     1   3.4E+003 3     1.2E+003  10606.3
6      SAH       9       5     0     1   2.8E+003 4     5.2E+002  6130.21
7      SAH       12      6     0     1   2.3E+003 4     3.3E+002  7098.42
8      SAFH      15      7     0     1   3.2E+003 4     3E+002    7480.99
9      SAFH      18      8     0     1   4.1E+003 4     3.2E+002  7649.91
10     SAH       19      9     0     1   1.5E+003 4     1.9E+002  7657.39
11     SAH       22      10    0     1   1.5E+003 4     1.3E+002  7964.39
12     SAH       26      11    0     1   2E+003   4     97        8154.14
13     SAH       30      12    0     1   1.3E+003 4     58        8287.43
14     SAH       33      13    0     1   7.4E+002 4     29        8405.27
...
88     SAH       129     87    0     1   3.9      2     0.0013    8566.11
89     SAH       130     88    0     1   4.7      2     0.0011    8566.11
90     SAH       131     89    0     1   4.5      2     0.001     8566.11
91     SAH       132     90    0     1   3.5      1     0.001     8566.11
92     SAFI      133     91    41    1   2.7      0     0.00075   8566.12
92     FOUND SOLUTION:   8566.12            in 3 sec.

93     SAI       134     92    35    1   4.3E-009 0     0.00075   8566.12
...
107    SAJ       144     102   0     0.1 0        0     0.00075   8566.12
108    AJ        144     102   0     0   0        0     0.00075   8566.12
108              Pointusage     5/90     Cutusage    11/341   (    0,133  )
109    AH        23      17    0     0   4.1      1     0.0011    8566.11
110    AIJ       24      18    0     0   3        0     0.00075   8566.12
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      : 8566.1150498670522
Max constraint (4)        : 0.00075412533010421612
Time used (seconds)       : 3.39
Time limit (seconds)      : 1000
Iterations used           : 110
Iteration limit           : 0
Function evaluations      : 1724
Gradient evaluations      : 359
Domain violations         : 0
Gradients unusable        : 0
Alphamax bound violations : 0
ECP time usage            : 2.8 %
NLP time usage            : 2.7 %
MIP time usage            : 94.5 %
Optimal/total MIPs        : 3/110
NLP solver calls          : 7
-------------------------------------------------------------------------------
```

In every iteration, information of the MIP problem and the modifications to the MIP problem, 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 underestimatos 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 *epsilon_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 evalauated.
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: MIP objective variable value.
NLobjval: MIPobjval is replaced with the nonlinear objective function value, **NLobjval**, when *ECPstrategy*

3) is used.

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 can not be found twice. When the cut reduction heuristic is called a printout is given, here is an example:

```
Pointusage     5/90     Cutusage    11/341    (    0,133  )
```

`Pointusage` informs how many points of all usable points have been used to generate the cutting planes. `Cutusage` tells how many cuts of all usable cuts have been used. The first number in `( 0,133 )` 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.
At the end of each solve AlphaECP gives a summary with Problem, Solver Status, Model Status, etc. Note the following lines **Exit comment, Domain violations, Gradients unusable** and **Alphamax bound violations**. **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** informs how many times a 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 use **ModelName.OptFile = 1**. The name of the option file is alphaecp.opt, see further information from the GAMS manual.

The following information is worth to notice when you are interested of 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. Recall that already one non-linear equality constraint makes a problem non-pseudoconvex, hence also non-convex. The basic options might impact significantly on 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 GAMS/AlphaECP Options

4.1 Basic options

CUTnrcuts	Cut generation pace
MIPnrsols	Upper limit of considered MIP solutions per MIP call
MIPsolstrat	MIP solution collection strategy
MIPsolver	MIP solver for subproblems
NLPsolver	NLP solver for subproblems
reslim	Time limit for AlphaECP (in seconds)

4.2 Algorithmic options for advanced users

CUTdelcrit	Cutting plane strategy
ECPbeta	Updating multiplier if MIP is infeasible

ECPdumpsol	Write encountered solutions to gdx files
ECPgamma	Updating multiplier when verifying solution
ECPiterlim	Maximum number of AlphaECP iterations
ECPloglevel	Level of AlphaECP output to statusfile
ECPpcostrategy	Pseudo-convex objective function strategy
ECPretsol	Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE)
ECPstart	User specified startpoint
ECPstrategy	AlphaECP strategy
solvelink	Solvelink for NLP and MIP subsolver
TOLepsf	Pseudo-convex objective function termination tolerance
TOLepsg	Constraint tolerance
TOLepsz	Distance tolerance for a new linearization
TOLgrad	Gradient tolerance
TOLinfbnd	Infinity bound (MIP variable bound)

4.3 MIP Solver related options

MIPloglevel	Level of MIP solver output
MIPoptcr	Relative mip gap in intermediate subproblems
MIPoptcrlim	Initial MIPoptcr interval before MIPoptcr reduction
MIPoptfile	Option file for MIP subsolver

4.4 NLP Solver related options

NLPcall	NLP strategy
NLPcalliter	NLP solver call at next (incremental) iteration
NLPlimsameint	NLP call after a number of recurring integer solutions
NLPloglevel	Level of NLP solver output
NLPreslim	NLP time limit per call

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, 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 in an iteration can be chosen by AlphaECP, proportional to the number of violating constraints or 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 in 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, then 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.

(default = 0)

0 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).

3 Report main actions at linearization 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).

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 startpoint

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

(default = 3)

0 Do not use a startpoint, start the algorithm by solving the linear part (MIP) of the problem.

1 Use the user specified startpoint, but the variable levels are adjusted with a small value.

2 Use the exact startpoint set by the user.

3 Use the exact startpoint 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.

2 Pseudo-convex constraints. Ensures global optimality for problems with convex objective function and pseudo-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. The reformulation can be done, assuming that the minimized or maximized variable is called objvar, 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 then the most suitable solution is chosen. The solutions from the MIP solver are many times 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 subproblems

The relative stopping tolerance which is sent to the MIP solver when solving the 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)

MIPoptfile *(integer)* Option file for MIP subsolver

By default the MIP subsolver is called without an option file. This option allows the user to specify an option number and therefore an option file to be used for the MIP subsolver runs.

(default = 0)

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 furthermore require the solutions to fullfill MIPoptcr >=0.2.

4 Let AlphaECP decide.

MIPsolver *(string)* MIP solver for subproblems

This option allows the user to specify a GAMS MIP subsolver, for example, CPLEX, GUROBI, XPRESS, etc. If no option is supplied the current active default MIP solver is selected.

(default = GAMS MIP solver)

NLPcall *(integer)* NLP strategy

Strategy that determines 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 a 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 that is 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 subproblems

`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 subsolver

(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.

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

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 *ECPstrategy 1*.

What are good settings if the solution speed is essential?
Try *ECPstrategy 1, CUTnrcuts 0.25* and *CUTdelcrit 1* and try if using multiple threads for the MIP solver improves the solution speed. However the chance of not finding an feasible solution for a non-convex problem with nonlinear equality constraints is considerable.

What are good settings when the solution quality is essential?
Try *MIPsolstrat 4* or *3*, *NLPcalliter 1* and try different values for *CUTdelcrit* 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.

7 AlphaECP References

Kelley J. E. (1960). The Cutting Plane Method for Solving Convex Programs. Journal of SIAM, Vol. VIII, No. 4, 703-712.

Pörn R. and Westerlund T. (2000). A Cutting Plane method for Minimizing Pseudo-convex Functions in the Mixed Integer Case. Computers Chem. Engng, 24, 2655-2665.

Still C. and Westerlund T. (2001). Extended Cutting Plane Algorithm. Encyclopedia of Optimization, Floudas and Pardalos (eds.), Kluwer Academic Publishers.

Westerlund T. and Pettersson F. (1995). An Extended Cutting Plane Method for Solving Convex MINLP Problems. Computers Chem. Engng Sup., 19, 131-136.

Westerlund T., Skrifvars H., Harjunkoski I. and Pörn R. (1998). An Extended Cutting Plane Method for Solving a Class of Non-Convex MINLP Problems. Computers Chem. Engng, 22, 357-365.

Westerlund T. and Pörn R. (2002). Solving Pseudo-Convex Mixed Integer Optimization Problems by Cutting Plane Techniques. Optimization and Engineering, 3, 253-280.

BARON

Nick Sahinidis; Carnegie Mellon University, Department of Chemical Engineering, 5000 Forbes Avenue, Pittsburgh, PA 15213, `sahinidis@cmu.edu`

Mohit Tawarmalani; Purdue University, Krannert School of Management, West Lafayette, IN 47907, `mtawarma@purdue.edu`

11 May 2011

Contents

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 availability of finite lower and upper bounds on the variables and their 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 and duality techniques for reducing ranges of variables in the course of the algorithm.

Parts of the BARON software were created at the University of Illinois at Urbana-Champaign.

1.1 Licensing and software requirements

In order to use GAMS/BARON, users will need to have a GAMS/BARON license as well as a licensed GAMS linear programming (LP) solver. A licensed GAMS nonlinear programming (NLP) solver is optional and usually expedites convergence. Current valid LP solvers include CPLEX, MINOS, SNOPT, and XPRESS. Current valid NLP solvers are CONOPT, MINOS, and SNOPT. Hence, a minimal GAMS/BARON system requires any one of the CONOPT, CPLEX, MINOS, SNOPT, or XPRESS solvers.

By default, GAMS/BARON will attempt to use CPLEX as the LP solver and MINOS as the NLP solver. If the user does not have licenses for these solvers, then the user must use the options `LPSol` and `NLPSol` to specify another LP or NLP solver. GAMS/BARON can be used without a local NLP solver by setting `DoLocal` $= 0$ and `NumLoc` $= 0$. See §4 on the BARON options.

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 xxx=baron;
```

where `xxx` stands for `LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP`, or `MINLP`.

2 Model requirements

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

2.1 Variable and expression bounds

All nonlinear variables and expressions in the mathematical program to be solved must be bounded below and above by finite numbers. 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 is not sufficient to guarantee finite bounds on nonlinear expressions arising in the model.

For example, consider the term $1/x$ for $x \in [0, 1]$, 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's preprocessor 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 §4 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 a relative and absolute tolerance of 1e-5.

```
===========================================================================
 BARON version 9.2.2. Built: LNX Fri May 6 08:02:56 EDT 2011

 Reference:
 Tawarmalani, M. and N. V. Sahinidis, A polyhedral
 branch-and-cut approach to global optimization,
 Mathematical Programming, 103(2), 225-249, 2005.

 BARON is a product of The Optimization Firm, LLC.
 Parts of the BARON software were created at the
 University of Illinois at Urbana-Champaign.
===========================================================================
 Factorable Non-Linear Programming
 LP solver:  ILOG CPLEX
 NLP solver: MINOS
===========================================================================
 Starting solution is feasible with a value of    0.361767610000D+02
 Doing local search
 Preprocessing found feasible solution with value 0.125706576060D+01
 Solving bounding LP
 Starting multi-start local search
 Preprocessing found feasible solution with value 0.100209253056D+01
 Done with local search
===========================================================================
 We have space for 597657 nodes in the tree (in 96 MB memory)
===========================================================================
  Iteration    Open Nodes       Total Time    Lower Bound      Upper Bound
         1             1         000:00:00    0.100000D+01     0.100209D+01
*        1             1         000:00:00    0.100000D+01     0.100117D+01
         1             1         000:00:00    0.100000D+01     0.100117D+01
*       14+            4         000:00:00    0.100000D+01     0.100018D+01
*       23             7         000:00:00    0.100000D+01     0.100013D+01
*       28             5         000:00:00    0.100000D+01     0.100006D+01
*       55+            0         000:00:00    0.100000D+01     0.100000D+01
        55             0         000:00:00    0.100000D+01     0.100000D+01

 Cleaning up solution and calculating dual

                        *** Normal Completion ***

  LP subsolver time:        000:00:00,    in seconds:          0.01
  NLP subsolver time:       000:00:00,    in seconds:          0.01
  All other time:           000:00:00,    in seconds:          0.03

  Total time elapsed:       000:00:00,    in seconds:          0.05
       on parsing:          000:00:00,    in seconds:          0.00
       on preprocessing:    000:00:00,    in seconds:          0.00
       on navigating:       000:00:00,    in seconds:          0.02
       on relaxed:          000:00:00,    in seconds:          0.00
```

```
      on local:             000:00:00,    in seconds:          0.00
      on tightening:        000:00:00,    in seconds:          0.00
      on marginals:         000:00:00,    in seconds:          0.00
      on probing:           000:00:00,    in seconds:          0.01

  Total no. of BaR iterations:      55
  Best solution found at node:      55
  Max. no. of nodes in memory:       7

 All done with problem
===========================================================================
```

The solver first tests feasibility of the user-supplied starting point. This point is found to be feasible with an objective function value of 0.361767610000D+02. BARON subsequently does its own search and, eventually, finds a feasible solution with an objective of 0.100209253056D+01. It then reports that the supplied memory (default of 96 MB) provides enough space for storing up to 597657 branch-and-reduce nodes for this problem.

Then, the iteration log provides information every 1000 iterations or every 30 seconds, whichever comes first. Additionally, information is printed at the end of the root node, whenever a better feasible solution is found, and at the end of the search. A star (`*`) in the first position of a line indicates that a better feasible solution was found. 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:

Field	**Description**
`Itn. no.`	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.
`Open Nodes`	Number of open nodes in branch-and-reduce tree.
`Total Time`	Current elapsed resource time in seconds.
`Lower Bound`	Current lower bound on the model.
`Upper Bound`	Current upper bound on the model.

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).

3.2 Termination messages, model and solver statuses

Upon 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:

$$\texttt{nodeopt} = \begin{cases} -3 & \text{no feasible solution found,} \\ -2 & \text{the best solution found was the user-supplied,} \\ -1 & \text{the best solution was found during preprocessing,} \\ i & \text{the best solution was found in the } i\text{th node of the tree.} \end{cases}$$

In addition to reporting `nodeopt`, upon termination, BARON will issue one of the following statements:

- `*** Normal Completion ***`. This is the 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.

- `*** User did not provide appropriate variable bounds ***`. The user will need to read the BARON output (in file `sum.scr` in the `gamskeep` directory) for likely 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 possible for BARON to construct reliable relaxations. 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 make more memory available to BARON or change algorithmic options to reduce the size of the search tree and memory required for storage. The user can increase the amount of available memory by using the GAMS options `WorkFactor` or `WorkSpace`.
- `*** Max. Allowable BaR 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`.
- `*** Numerical Difficulties Encountered ***`. This case should be reported to the developers.
- `*** 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 can increase the amount of available memory by using the GAMS options `WorkFactor` or `WorkSpace`.
- `*** Search Terminated by BARON ***`. This will happen in certain cases when the required variable bounds are not provided in the input model. The user will need to read the BARON output for likely pointers to variables and expressions with missing bounds and fix the formulation, or be content with the solution provided. In the latter case the solution may not be globally optimal.

4 Some BARON features

The features described in these 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.

4.1 No starting point is required

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 `SOLVE` statement:

```
MaxExecError = 100000;
option sys12 = 1;
```

The first command asks GAMS to continue compilation for as many as `MaxExecError` execution errors. The `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.

[1] Tawarmalani, M. and N. V. Sahinidis, *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, 504 pages, Kluwer Academic Publishers, Dordrecht, Vol. 65 in *Nonconvex Optimization And Its Applications* series, 2002.

4.2 Finding the best, second best, third best, etc. solution, or all feasible solutions

BARON offers a facility, through its `NumSol` option to find the best few, or even all feasible, solutions to a model. Modelers interested in finding multiple solutions of integer programs often use integer cuts. The integer program is solved to optimality, an integer cut is added to the model in order to make the previous solution infeasible, and the model is solved again to find the second best integer solution. This process can then be repeated to obtain the best several solutions or all the feasible solutions of an integer program. This approach requires the user to explicitly model the integer cuts as part of the GAMS model.

In addition to eliminating the need for coding of integer cuts by the user, BARON does not rely on integer cuts to find multiple solutions. Instead, BARON directly eliminates a single search tree, which leads to a computationally more efficient method for finding multiple solutions. Furthermore, BARON's approach applies to integer as well as continuous programs. Hence, it can also 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 `NumSol` option, the solutions found can be recovered using the GAMS GDX facility. An example is provided below.

```
$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 /

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.scr';
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, LPTTDo, OBTTDo,` and `PreLPDo`. 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 `MaxPreTime` CPU seconds have elapsed.

5 The BARON options

BARON 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, OptFile`, and `CutOff`. The `IterLim` option is not implemented as it refers to simplex iterations and BARON specifies nodes. To specify BARON iterations, the user can set the `MaxIter` option, which is equivalent to the GAMS option `NodLim`. A description of GAMS options can be found in Chapter "Using Solver Specific Options."

If you specify "`<modelname>.optfile = 1;`" before the `SOLVE` statement in your GAMS model, BARON will then look for and read an option file with the name *baron.opt* (see "Using Solver Specific Options" for general use of solver option files). The syntax for the BARON option file is

```
   optname value
```

with one option on each line. For example,

```
  * This is a typical GAMS/BARON options file.
  * We will rely on the default BARON options with
  * two exceptions.
  pdo 3
  prfreq 1000
```

Lines beginning with `*` are considered comments and ignored. The first option specifies that probing will be used to reduce the bounds of three variables at every node of the tree. The second option specifies that log output is to be printed every 100 nodes.

The BARON options allow the user to control variable bounds and priorities, termination tolerances, branching and relaxation strategies, heuristic local search options, and output options as detailed next.

5.1 Setting variable bounds and branching priorities

Variable Bounds. BARON requires bounded variables and expressions to guarantee global optimality. The best way to provide such bounds is for the modeler to supply physically meaningful bounds for all problem variables using the `.lo` and `.up` variable attributes in the GAMS file. Alternatively, bounds may be provided to BARON in the form of *solver bounds* that are not be part of the user's model. To specify such solver bounds for BARON, create a BARON solver option file as described above. For lower and upper variable bounds the syntax is:

(variable)`.lo` *(value)*

(variable)`.up` *(value)*

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j);
```

Then, the BARON bounds in the *baron.opt* file can be specified by:

```
v.lo            0
v.up            1
v.lo('i1','j2') 0.25
v.up('i1',*)    0.5
```

We specify that all variables `v(i,j)` have lower bounds of `0` and upper bounds of `1`, except variables over set element `i1`, which have upper bound `0.5`. The variable over set element `i1` and `j2` has lower bound `0.25`. Note that variable bounds are assigned in a procedural fashion so that bounds assigned later overwrite previous bounds.

Consider also the following GAMS example for expression bounds:

```
v =E= log(z);
z =E= x-y;
```

where `x,y,v,z` are variables. In order to ensure feasibility, we must have `x>y`, guaranteeing `z>0`. We can specify expression bounds in BARON using the solver option file:

```
z.lo 0.00001
```

which is equivalent to specifying in GAMS that the expression `x-y =G= 0.00001` and thereby bounding `v`.

Variable Priorities. BARON implements branch-and-bound algorithms involving convex relaxations of the original problem. Branching takes place not only on discrete variables, but also on continuous ones that are nonlinear. Users can specify branching priorities for both discrete and continuous variables.

To specify variable branching priorities, one specifies

(variable)`.prior` *(value)*

in the *baron.opt* file, where *(value)* can be any positive (real) value. Default priorities are `1` for all variables, including continuous ones. The option `bpint` can be used to adjust the priorities of integer variables. Priorities of integer variables are multiplied by `bpint`. By default, this option has a value of 1, thus placing equal emphasis on integer and continuous variables.

BARON priorities are assigned in a manner such that a larger value implies a higher priority. In contrast, GAMS priorities are assigned in such a fashion that a larger value implies a lower priority. BARON and GAMS variable priorities are related by

$$BARON\ priority = 1/GAMS\ priority$$

5.2 Termination options

Option	Description	Default
`EpsA` (ϵ_a)	Absolute termination tolerance. BARON terminates if $U - L \leq \epsilon_a$, where U and L are the lower and upper bounds to the optimization problem at the current iteration. This is equivalent to the GAMS option `OptCA`.	`1e-9`
`EpsR` (ϵ_r)	Relative termination tolerance. BARON terminates if $L > \infty$ and $U - L \leq \epsilon_r\|L\|$, where U and L are the lower and upper bounds to the optimization problem at the current iteration. This is equivalent to the GAMS option `OptCR`.	`0.1`
`ConTol`	Constraint satisfaction tolerance.	`1e-5`
`BoxTol`	Box elimination tolerance.	`1e-8`
`IntTol`	Integrality satisfaction tolerance.	`1e-6`
`FirstFeas`	If set to 1, BARON will terminate once it finds `NumSol` feasible solutions, irrespective of solution quality. By default, `FirstFeas` is 0, meaning that BARON will search for the *best* `NumSol` feasible solutions.	`0`
`MaxIter`	Maximum number of branch-and-reduce iterations allowed. -1 implies unlimited. This is equivalent to the GAMS option `NodLim`. 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.	`-1`
`MaxPreTime`	Maximum CPU time allowed (sec) to be spent in preprocessing. If set to -1, the `MaxTime` limit apply.	`-1`
`MaxTime`	Maximum CPU time allowed (sec). This is equivalent to the GAMS option `ResLim`. If unspecified, the GAMS resource limit is enforced.	`1200`
`NumSol`	Number of feasible solutions to be found. Solutions found will be listed in the `res.scr` file in the `gamskeep` directory. 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 `res.scr`.	`1`
`IsolTol`	Separation distance between solutions. This option is used in conjunction with `NumSol`. For combinatorial optimization problems, feasible solutions are isolated. For continuous problems, feasible solutions points within an l_∞ distance that does not exceed `IsolTol` $>$ `0` will be treated as identical by BARON.	`1e-4`

5.3 Relaxation options

Option	Description	Default
`NLPDoLin`	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.	`1`
`linearidentify`	Identification of common linear subexpressions of nonlinear functions is done by default during relaxation construction; it can be turned off by using a value of 0 for this option. The default value may result in tighter relaxations but some models may require a large time during BARON's parsing and reformulation stage when this option is in effect.	`1`
`MultMSize`	Size of maximum allowable multilinear function for cutting plane generation; larger multilinear functions are decomposed to multilinears of size no more than this parameter.	`7`
`MultRel`	Number of rounds of cutting plane generation from envelopes of multilinear functions at LP relaxation.	`1`
`nOuter1`	Number of outer approximators of convex univariate functions.	`4`
`NOutPerVar`	Number of outer approximators per variable for convex multivariate functions.	`4`
`NOutIter`	Number of rounds of cutting plane generation at LP relaxation.	`4`
`OutGrid`	Number of grid points per variable for convex multivariate approximators.	`20`

5.4 Range reduction options

Option	Description	Default
`TDo`	Nonlinear-feasibility-based range reduction option (poor man's NLPs). `0`: no bounds tightening is performed. `1`: bounds tightening is performed.	`1`
`MDo`	Marginals-based reduction option. `0`: no range reduction based on marginals. `1`: range reduction done based on marginals.	`1`
`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`
`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. `0`: no range reduction by probing. `-1`: probing on all `NumBranch` variables. n: probing on n variables.	`3`
`PBin`	Probing on binary variables option. `0`: no probing on binary variables. `1`: probing on binary variables.	`0`
`PXDo`	Number of probing variables fully optimized (not fixed at bound).	`-1`
`PStart`	Level of branch-and-reduce tree where probing begins. `0`: probing begins at root node. n: probing begins at level n.	`0`
`PEnd`	Level of branch-and-reduce tree where probing ends. `-1`: probing never ends. n: probing ends at level n.	`-1`
`PFreq`	Level-frequency of probing applications. `1`: probing is done at every level of the search tree. n: probing is done every n levels, beginning at level `PStart` and ending at level `PEnd`.	`3`
`ProFra`	Fraction of probe to bound distance from relaxed solution when forced probing is done.	`0.67`
`TwoWays`	Determines wether probing on both bounds is done or not. `0`: probing to be done by farthest bound `1`: probing to be done at both bounds	`1`
`MaxRedPass`	Maximum number of times range reduction is performed at a node before a new relaxation is constructed. At any given node, at most `MaxRedPass` calls of the range reduction heuristics will be performed for tightening based on feasibility, marginals, and probing in accordance to the options `TDo`, `MDo`, and `PDo`, respectively.	`10`
`MaxNodePass`	Maximum number of passes (relaxation constructions) allowed through a node. If postprocessing improves the node's lower bound in a way that satisfies the absolute or relative tolerances, `RedAbsTol` or `RedRelTol`, respectively, the process of lower bounding followed by postprocessing is repeated up to `MaxNodePass` times.	`5`
`RedRelTol`	Relative improvement in the objective to reconstruct the relaxation of the current node.	`0.1`
`RedAbsTol`	Absolute improvement in the objective to reconstruct the relaxation of the current node.	`0.1`

5.5 Branching options

Option	Description	Default
`BrVarStra`	Branching variable selection strategy. `0:` BARON's dynamic strategy `1:` largest violation `2:` longest edge	`0`
`BrPtStra`	Branching point selection strategy. `0:` BARON's dynamic strategy `1:` ω-branching `2:` bisection-branching `3:` convex combination of ω and bisection as dictated by `ConvexRatio`	`0`
`ConvexRatio`	The branching point under `BrPtStra` $= 3$ is set to `ConvexRatio` $* \omega + (1 -$ `ConvexRatio`$) * \beta$, where ω and β are the ω- and bisection-branching points.	`0.7`
`ModBrpt`	Branch point modification option. `0:` `BrPtStra`-dictated branching point is used without any modifications. `1:` allows BARON to occasionally modify the `BrPtStra`-dictated branching point, if deemed necessary.	`1`
`NumBranch`	Number of variables to be branched on. `-1:` consider the model variables as well as variables introduced by BARON's lower bounding procedure. `0:` consider only the original model variables for branching. n`:` consider only the first n variables for branching. This option requires knowledge about variable orders and is recommended for *advanced users only.*	`0`
`NumStore`	Number of variables whose bounds are to be stored at every node of the tree. `0:` store `NumBranch` variables `-1:` store all variables n`:` store n variables This option requires knowledge about variable orders and is recommended for *advanced users only.*	`0`

5.6 Heuristic local search options

Option	Description	Default
DoLocal	Local search option for upper bounding. 0: no local search is done during upper bounding 1: BARON's dynamic local search decision rule $-n$: local search is done once every n iterations	1
MaxHeur	Maximum number of passes allowed for local search heuristic, provided the upper bound improvement during two consecutive passes satisfies either the relative or absolute improvement tolerance (see HRelTol and HAbsTol).	5
HabsTol	Absolute improvement requirement in the objective for continuation of local search heuristic.	0.1
HRelTol	Relative improvement requirement in the objective for continuation of local search heuristic.	0.1
NumLoc	Number of local searches done in NLP preprocessing. The first one begins with the user-specified starting point as long as it is feasible. Subsequent local searches are done from judiciously chosen random starting points. If NumLoc is set to -1, local searches in preprocessing will be done until proof of globality or MaxPreTime is reached.	10
LocRes	Option to control output to log from local search. 0: no local search output. 1: detailed results from local search will be printed to res.scr file.	0

5.7 Output options

Option	Description	Default
PrFreq	Log output frequency in number of nodes.	1000
PrTimeFreq	Log output frequency in number of seconds.	30
PrLevel	Level of results printed. A larger value produces more output. ≤ 0: all log output is suppressed > 0: print log output	1
DotBar	Name of BARON problem file to be written.	
ObjName	Name of objective variable to be optimized. By default, BARON writes the complete objective function to be optimized. If the user specifies an ObjName, this will be written in place of an objective function in the DotBar file, provided a Reform level of 0 is used. Useful only in conjunction with DotBar and if Reform is set to 0.	
Reform	Reformulation level of problem. A value of 0 indicates no reformulation: the complete objective function is listed as an additional constraint and the model minimizes an objective variable. A value of 1 replaces the objective variable by the objective constraint. This is sometimes useful for reducing the model size. A larger Reform value indicates a more aggressive reformulation (if possible).	100

5.8 Other options

Option	Description	Default
`eqname.equclass`	Specifies nature of constraint named `eqname` in the user's model. Slices like "`supply.equclass('new-york') 1`" are allowed. `0:` Regular constraint. `1:` Relaxation-only constraint. These constraints are provided to BARON as `RELAXATION_ONLY_EQUATIONS` and used to help strengthen the relaxation bound but are not considered as part of the user model and thus not used for feasibility testing of solutions or local search. Adding, for instance, the first-order optimality conditions as relaxation-only constraints often expedites convergence. `2:` Convex constraint. These constraints are provided to BARON as `CONVEX_EQUATIONS` and used to generate cutting planes from the set of outer approximating supporting hyperplanes of the convex constraint set. `3:` Convex constraint that is relaxation-only.	`0`
`LPSol`	Specifies the LP solver to be used. `2:` MINOS `3:` CPLEX `4:` SNOPT `7:` XPRESS	`3`
`LPAlg`	Specifies the LP algorithm to be used (available only with CPLEX and XPRESS as the LP solver). `0:` automatic selection of LP algorithm `1:` primal simplex `2:` dual simplex `3:` barrier	`0`
`NLPSol`	Specifies the NLP solver to be used. `2:` MINOS `4:` SNOPT `6:` GAMS NLP solver (see ExtNLPsolver)	`2`
`ExtNLPSolver`	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.	`CONOPT`
`BasKp`	Indicates whether basis information is to be saved. `0:` no basis information is saved `1:` LP solver working basis will not be modified if at least `basfra` $* n$ of its basic variables are also basic in the saved basis for the node that is about to be solved.	`1`
`BasFra`	Similarity measure between bases for basis update not to occur.	`0.7`
`InfBnd`	Infinity value to be used for variable bounds. If set to 0, then no bounds are used.	`0`
`NodeSel`	Specifies the node selection rule to be used for exploring the search tree. `0:` BARON's `1:` best bound `2:` LIFO `3:` minimum infeasibilities	`0`

Option	**Description**	**Default**
`PostAbsTol`	Absolute tolerance for postponing a node. See `PostRelTol`.	`1e30`
`PostRelTol`	Relative tolerance for postponing a node. Instead of branching after solving a node, it is often advantageous to postpone the current node if its lower bound is sufficiently above the (previously) second best lower bound in the branch-and-bound tree. Let z and $z2$ denote the current node's lower bound and the previously second best lower bound in the branch-and-bound tree, respectively. Postponement of a node will take place if any of the following two conditions holds: • $z - z2 \geq$ `PostAbsTol` • $z - z2 \geq$ `PostRelTol` $\times \lvert z2 \rvert$	`1e30`
`PreLPDo`	Number of preprocessing LPs to be solved in preprocessing. -n: preprocess the first n problem variables `0`: no preprocessing LPs should be solved `1`: preprocess all problem variables including those introduced by BARON's reformulator `2`: preprocess the first `NumStore` problem variables `3`: preprocess all original problem variables	`1`
`CutOff`	Ignore solutions that are no better than this value. Can also be used as GAMS model suffix option: *(modelname).cutoff = (value).*	∞

BDMLP

Contents

1 Introduction

GAMS/BDMLP is an LP and MIP solver that comes 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. The MIP part was added by M. Bussieck and A. Drud. GAMS/BDMLP is running 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 types of discrete variables supported by GAMS: Binary, Integer, Semicont, 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:

```
option lp = bdmlp; { or RMIP or MIP }
```

BENCH

Contents

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.

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 §2.

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 "Using Solver Specific Options" 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.

Option	**Description**	**Default**
`iterlim`	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.	`10000`
`reslim`	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.	`1000`

2.2 The BENCH Options

BENCH solver options are passed on through solver option files. If you specify "`<modelname>.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 "Using Solver Specific Options" 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. See the section describing the BENCH listing file for details.

Option	**Description**	**Default**
`allsolvers`	Indicator if all valid solvers for given modeltype should be run.	`0`
`cumulative`	Indicator if resource time and iteration limits are interpreted as total limits for all solvers or for individual solvers. For example if enabled and `reslim=1500` then all solvers have a total of `1500` seconds. If not enabled, then each solver has a limit of `1500` seconds. `0:` limits enforced for each solver separately `1:` limits enforced cumulatively	`0`
`dualcstol`	EXAMINER `dualcstol` option. Dual complementary slackness tolerance. By dual CS we refer to complementary slackness between dual variables and the primal constraints.	`1e-7`
`dualfeastol`	EXAMINER `dualfeastol` option. Dual feasibility tolerance. This tolerance is used for the checks on the dual variables and the dual constraints.	`1e-6`
`examiner`	Flag to call GAMS/EXAMINER to independently verify solution for feasibility and optimality. `0:` skip solution verification. `1:` verify solution using EXAMINER.	`0`
`outlev`	Log output level. `1:` BENCH summary log output only. `2:` BENCH summary log output and log output of each solver.	`2`
`paver`	Indicator if PAVER trace files should be written. Enabling causes a trace file `solver.pvr` to be written for each solver called. If the solver uses an option file, then the resulting file is `solver-optnum.pvr`, where `optnum` is the option file number. The files created can be submitted to the PAVER Server for automated performance analysis. See `http://www.gamsworld.org/performance/paver` `0:` no PAVER trace files written `1:` write trace file for each solver called	`0`
`paverex`	Indicator if PAVER trace files should be written for the Examiner run. Enabling causes a trace file `solver-ex.pvr` 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 `solver-optnum-ex.pvr`, where `optnum` is the option file number. The files created can be submitted to the PAVER Server for automated performance analysis. See `http://www.gamsworld.org/performance/paver` `0:` no PAVER trace files for Examiner run written `1:` write trace file for each solver called	`0`
`primalcstol`	EXAMINER `primalcstol` option. Primal complementary slackness tolerance. By primal CS we refer to complementary slackness between primal variables and the dual constraints.	`1e-7`
`primalfeastol`	EXAMINER `primalfeastol` option. Primal feasibility tolerance. This tolerance is used for the checks on the primal variables and the primal constraints.	`1e-6`
`returnlastsol`	Return the solution information of the last solver. `0:` Do not return a solution. `1:` Return the solution of last solver.	`0`

Option	Description	Default
`solvers`	`solver[.n] Solver` is the name of the GAMS solver that should be used, and `n` is the integer corresponding to optfile for the root node. If `.n` is missing, the optfile treated as zero (i.e., the solver) will not look for an options file. This is a required option.	`none`

2.3 Solvers Requiring Subsolvers

For GAMS solvers requiring subsolvers, for example the MINLP solvers DICOPT and SBB, BENCH requires some care. By default, BENCH assumes the default subsolvers for solvers requiring them. For example, if an MINLP model is solved via BENCH, then the MINLP solvers specified by the user in the `bench.opt` solver option file only call the default subsolvers available to them and *do not* run all valid subsolvers.

If users wish to do benchmarks with particular subsolvers, then subsolvers can be passed along via subsolver option files. To tell BENCH to use a particular solver, users can use the `solvers` option in the BENCH option file. For example, the BENCH option

```
solvers sbb dicopt
```

specifies that the solvers SBB and DICOPT are executed (and in that particular order). The subsolvers used by SBB and DICOPT will be the default solvers for your particular GAMS system.

To use a particular subsolver, users can append to the solver indicator. For example

```
solvers sbb sbb.1 sbb.2 dicopt dicopt.1
```

specifes that SBB without an option file is called first, then SBB with an option file called *sbb.opt* and then with an option file called *sbb.op2*. Then DICOPT is called without an option file and then with an option file called `dicopt.op2`. Within these option files, the particular subsolvers can be specified. The input of the solver option file is echoed in the listing file created by BENCH to help distinguish the different solver calls.

3 Benchmark Analysis Using the PAVER Server

Benchmark data obtained using GAMS/BENCH can be automatically analyzed using the PAVER Server. For more information on PAVER, see `http://www.gamsworld.org/performance/paver`.

In order to enable creation of the necessary data files for submission to PAVER, users must enable the `paver` option as described in §2.

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[1] 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 subsolver. The subsolver 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 variabless) 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. See the BENCH solver options for details. 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 (`solvpoint`). 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 `trnsport.gms`:

```
Solver     Modstat  Solstat    Objective       ResUsd    Examiner
-----------------------------------------------------------------
BDMLP           1        1      153.6750        0.000       P/P
MINOS           1        1      153.6750        0.000       P/P
XPRESS          1        1      153.6750        0.040       P/P
CPLEX           1        1      153.6750        0.000       P/P
```

[1] See `http://www.gams.com/solvers/examiner.pdf`

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`).

```
Solver     Modstat  Solstat    Objective       ResUsd    Examiner
-----------------------------------------------------------------
BDMLP          5       1          0.0000         0.000       N/N
```

For models having discrete variables, for example MIP, MIQCP, or MINLP, we also show the best bound. A sample output using the model `magic.gms` is shown below.

```
Solver    Modstat Solstat      Objective      BestBound      ResUsd  Examiner
-----------------------------------------------------------------------------
CPLEX           8       1    991970.0000    985514.2857       0.000      n/a
XPRESS          1       1    988540.0000    988540.0000       0.060      n/a
MOSEK           8       1    988540.0000    988540.0000       0.170      n/a
```

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 (`trnsport.gms`) from the GAMS model library. We specify the solvers `BDMLP`, `XPRESS`, `MINOS` and `CPLEX` via the option file `bench.opt`:

```
GAMS Rev 138  Copyright (C) 1987-2004 GAMS Development. All rights reserved
Licensee: GAMS Development Corp.                         G040421:1523CR-LNX
          GAMS Development Corp.                                     DC3665
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution
--- trnsport.gms(45) 4 Mb
--- Generating model transport
--- trnsport.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

    Iter    Sinf/Objective  Status    Num  Freq
       1    2.25000000E+02  infeas      1     1
       4    1.53675000E+02  nopt        0
 SOLVER STATUS: 1 NORMAL COMPLETION
 MODEL STATUS : 1 OPTIMAL
 OBJECTIVE VALUE     153.67500

--- Spawning solver : MINOS


MINOS-Link    Jan 19, 2004 LNX.M5.M5 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     Jan 19, 2004 LNX.XP.XP 21.3 024.027.041.LXI Xpress lib 14.24
Xpress-MP licensed by Dash to GAMS Development Corp. for GAMS

Reading data . . . done.

Reading Problem gmsxp_xx
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

   Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
     0           .000000      D      3     0     900.000000     0
     3        153.675000      D      0     0        .000000     0
Uncrunching matrix
     3        153.675000      D      0     0        .000000     0
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.

Iteration      Dual Objective            In Variable           Out Variable
     1             73.125000    x(seattle.new-york) demand(new-york) slack
     2            119.025000     x(seattle.chicago)  demand(chicago) slack
     3            153.675000    x(san-diego.topeka)   demand(topeka) slack
     4            153.675000  x(san-diego.new-york)  supply(seattle) slack

Optimal solution found.
Objective :         153.675000



--- BENCH SUMMARY:

  Solver     Modstat  Solstat    Objective       ResUsd    Examiner
  -----------------------------------------------------------------
  BDMLP          1       1        153.6750        0.000       n/a
  MINOS          1       1        153.6750        0.000       n/a
  XPRESS         1       1        153.6750        0.040       n/a
  CPLEX          1       1        153.6750        0.000       n/a

--- 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 `batchdes.gms` 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:

```
solvers sbb.1 dicopt
```

Note that SBB will use a solver option file called *sbb.opt*.

```
               S O L V E     S U M M A R Y

   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. See the BENCH solver option section.

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.

```
                 B E N C H M A R K     S U M M A R Y


       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 Ctl-C

BENCH passes all *Control-C* (Ctl-C) signals to the respective subsolvers. If a terminate signal via Ctl-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 Ctl-C signal should be sent at the beginning of a solver's execution. Thus, several Ctl-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.

Firs 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 for automated performance analysis. Now create an option file `conopt3.opt` with the entry

```
lsscal 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 iterlim=999999 domlim=99999
$call gams catmix.gms   nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams chain.gms    nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams elec.gms     nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
```

```
$call gams flowchan.gms nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams gasoil.gms   nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams glider.gms   nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams jbearing.gms nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams lnts.gms     nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams methanol.gms nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams minsurf.gms  nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams pinene.gms   nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams polygon.gms  nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams popdynm.gms  nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams robot.gms    nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams rocket.gms   nlp=bench optfile=1 reslim=10 iterlim=999999 domlim=99999
$call gams torsion.gms  nlp=bench optfile=1 reslim=10 iterlim=999999 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 Figure 5.1 shows the efficiency of each solver/solver option.

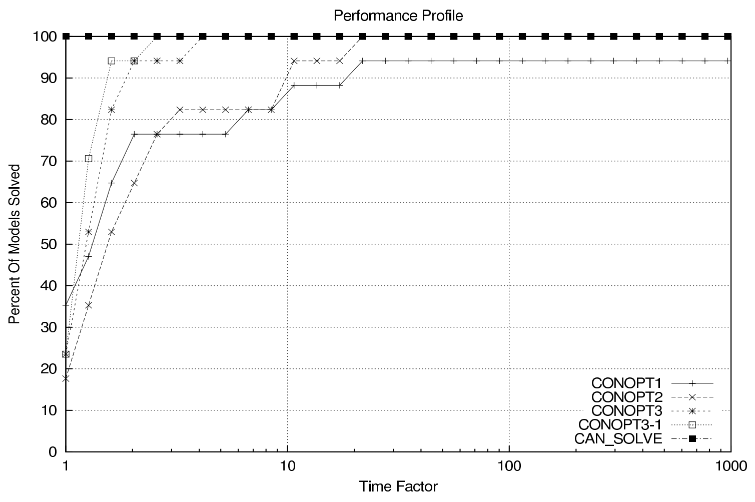


Figure 5.1: PAVER: Process Overview

COIN-OR

Stefan Vigerske, Humboldt University Berlin, Germany

Contents

1 Introduction

COIN-OR (**CO**mputational **IN**frastructure for **O**perations **R**esearch, `http://www.coin-or.org`) is an initiative to spur the development of open-source software for the operations research community [19]. One of the projects hosted at COIN-OR is the GAMSlinks project (`https://projects.coin-or.org/GAMSlinks`). It is dedicated to the development of interfaces between GAMS and open source solvers. Some of these links and solvers have also found their way into the regular GAMS distribution. With the availability of source code for the GAMSlinks the user is not limited to the out of the box solvers that come with a regular GAMS distribution, but can extend and build these interfaces by themselves.

Available solvers and tools include:

- Bonmin: Basic Open-source Nonlinear Mixed Integer programming
(model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- CBC: COIN-OR Branch and Cut
(model types: LP, RMIP, MIP)
- Couenne: Convex Over and Under Envelopes for Nonlinear Estimation
(model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- Ipopt: Interior Point Optimizer
(model types: LP, RMIP, DNLP, NLP, RMINLP, QCP, RMIQCP)
- OS: Optimization Services
(model types: LP, RMIP, MIP, DNLP, NLP, RMINLP, MINLP, QCP, RMIQCP, MIQCP)
- OsiCplex, OsiGlpk, OsiGurobi, OsiMosek, OsiSoplex, OsiXpress: Open Solver Interface
(model types: LP, RMIP, MIP)

For more information see the COIN-OR/GAMSlinks web site at `https://projects.coin-or.org/GAMSlinks`.

2 Bonmin

Bonmin (**B**asic **O**pen-source **N**onlinear **M**ixed **In**teger programming) is an open-source solver for mixed-integer nonlinear programming (MINLPs). 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 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 [17]; 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 [12, 13]; 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 [21].
- 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 [11]
- 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.

For more information we refer to [8, 9, 11, 7] and the BONMIN web site `https://projects.coin-or.org/Bonmin`. Most of the BONMIN documentation in this section is taken from the BONMIN manual [10].

2.1 Model requirements

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).

If GAMS/BONMIN is called for a model with only continuous variables, the interface switches over to IPOPT. If GAMS/BONMIN is called for a model with only linear equations, the interface switches over to CBC.

2.2 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.

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 $\leq$ 23.3, available at `http://www.gams.com/download/download_old.htm`.

Specification of Options

A BONMIN option 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. Specifying `optfile=1` let GAMS/BONMIN read `bonmin.opt`, `optfile=2` corresponds to `bonmin.op2`, and so on. The format of the option file is the same as for IPOPT (see Section 5.2).

The most important option in BONMIN is the choice of the solution algorithm. This can be set by using the option named `bonmin.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 2.3.

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: `reslim` (time limit), `iterlim` (iteration limit), `nodlim` (node limit), `cutoff`, `optca` (absolute gap tolerance), and `optcr` (relative gap tolerance). One can set them either on the command line, e.g. `nodlim=1000`, or inside your GAMS program, e.g. `Option nodlim=1000;`. Further, under Linux and Windows, the option `threads` can be used to control the number of threads used in the linear algebra routines of IPOPT.

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 "`bonmin.`" prefix is replaced with a prefix that identifies the sub-algorithm used:

- to pass options to Outer Approximation Decomposition: `oa_decomposition.`,
- to pass options to Feasibility Pump for MINLP: `pump_for_minlp.`,
- to pass options to RINS: `rins.`,
- to pass options to RENS: `rens.`,
- to pass options to Local Branching: `local_branch`.

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 `bonmin.milp_solver`. Values for this option are: `Cbc_D` which uses CBC with its default settings, `Cbc_Par` which uses a version of CBC that can be parameterized by the user, and `Cplex` which uses CPLEX with its default settings. The options that can be set in `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, c.f. Section 2.3 for details. To use the `Cplex` option, a valid CPLEX licence (standalone or GAMS/CPLEX) is required.

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 naïve: 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` and `allowable_fraction_gap` and `cutoff_decr` to negative values.

Ipopt options changed by Bonmin

Ipopt has a very large number of options, see Section 5.4 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 10^{-8} and 10^{-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 `nlp_log_at_root`).
- `bound_relax_factor` is set to 10^{-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.

2.3 Detailed Options Description

The following tables gives the list 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 MLIP subsolver in the context of OA and FP.

Table 6.1: List of options and compatibility with the different algorithms.

Option	type	default	`B-BB`	`B-OA`	`B-QG`	`B-Hyb`	`B-Ecp`	`B-iFP`	`Cbc_Par`
		Algorithm choice							
algorithm	string	B-BB	✓	✓	✓	✓	✓	✓	✓
		Branch-and-bound options							
allowable_fraction_gap	$\mathbb{Q}$	GAMS `optcr`	✓	✓	✓	✓	✓	✓	✓
allowable_gap	$\mathbb{Q}$	GAMS `optca`	✓	✓	✓	✓	✓	✓	✓

continued on next page

Option	type	default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Par
cutoff	$\mathbb{Q}$	GAMS `cutoff`	✓	✓	✓	✓	✓	✓	✓
cutoff_decr	$\mathbb{Q}$	10^{-5}	✓	✓	✓	✓	✓	✓	✓
enable_dynamic_nlp	string	no	–	–	✓	✓	✓	–	–
integer_tolerance	$\mathbb{Q}$	10^{-6}	✓	✓	✓	✓	✓	✓	✓
iteration_limit	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	✓
nlp_failure_behavior	string	stop	✓	–	–	–	–	–	–
node_comparison	string	best-bound	✓	✓	✓	✓	✓	✓	–
node_limit	$\mathbb{Z}$	GAMS `nodlim`	✓	✓	✓	✓	✓	✓	✓
num_cut_passes	$\mathbb{Z}$	1	–	–	✓	✓	✓	–	–
num_cut_passes_at_root	$\mathbb{Z}$	20	–	–	✓	✓	✓	–	–
number_before_trust	$\mathbb{Z}$	8	✓	✓	✓	✓	✓	✓	✓
number_strong_branch	$\mathbb{Z}$	20	✓	✓	✓	✓	✓	✓	✓
solution_limit	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	✓
time_limit	$\mathbb{Q}$	GAMS `reslim`	✓	✓	✓	✓	✓	✓	✓
tree_search_strategy	string	probed-dive	✓	✓	✓	✓	✓	✓	–
variable_selection	string	strong-branching	✓	–	–	–	–	–	–
ECP cuts generation									
ecp_abs_tol	$\mathbb{Q}$	10^{-6}	–	–	✓	✓	–	–	–
ecp_max_rounds	$\mathbb{Z}$	5	–	–	✓	✓	–	–	–
ecp_probability_factor	$\mathbb{Q}$	10	–	–	✓	✓	–	–	–
ecp_rel_tol	$\mathbb{Q}$	0	–	–	✓	✓	–	–	–
filmint_ecp_cuts	$\mathbb{Z}$	0	–	–	✓	✓	–	–	–
Feasibility checker using OA cuts									
feas_check_cut_types	string	outer-approx	–	–	✓	✓	✓	–	–
feas_check_discard_policy	string	detect-cycles	–	–	✓	✓	✓	–	–
generate_benders_after_so_many_oa	$\mathbb{Z}$	5000	–	–	✓	✓	✓	–	–
MILP Solver									
cpx_parallel_strategy	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
milp_solver	string	Cbc_D	–	–	–	–	–	–	✓
milp_strategy	string	find_good_sol	–	–	–	–	–	–	✓
number_cpx_threads	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
MILP cutting planes in hybrid algorithm (B-Hyb)									
2mir_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
Gomory_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
clique_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
cover_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
flow_cover_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
lift_and_project_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
mir_cuts	$\mathbb{Z}$	-5	–	✓	✓	✓	✓	✓	✓
reduce_and_split_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
MINLP Heuristics									
feasibility_pump_objective_norm	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	–
fp_pass_infeasible	string	no	✓	✓	✓	✓	✓	✓	✓
heuristic_RINS	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_MIP_fractional	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_MIP_vectorLength	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_fractional	string	no	✓	✓	✓	✓	✓	✓	–

continued on next page

Option	type	default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Par
heuristic_dive_vectorLength	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_feasibility_pump	string	no	✓	✓	✓	✓	✓	✓	–
pump_for_minlp	string	no	✓	✓	✓	✓	✓	✓	–
NLP interface									
warm_start	string	none	✓	–	–	–	–	–	–
NLP solution robustness									
max_consecutive_failures	$\mathbb{Z}$	10	✓	–	–	–	–	–	–
max_random_point_radius	$\mathbb{Q}$	100000	✓	–	–	–	–	–	–
num_iterations_suspect	$\mathbb{Z}$	-1	✓	✓	✓	✓	✓	✓	✓
num_retry_unsolved_random_point	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	✓
random_point_perturbation_interval	$\mathbb{Q}$	1	✓	–	–	–	–	–	–
random_point_type	string	Jon	✓	–	–	–	–	–	–
NLP solves in hybrid algorithm (B-Hyb)									
nlp_solve_frequency	$\mathbb{Z}$	10	–	–	–	✓	–	–	–
nlp_solve_max_depth	$\mathbb{Z}$	10	–	–	–	✓	–	–	–
nlp_solves_per_depth	$\mathbb{Q}$	10^{100}	–	–	–	✓	–	–	–
Nonconvex problems									
coeff_var_threshold	$\mathbb{Q}$	0.1	✓	–	–	–	–	–	–
dynamic_def_cutoff_decr	string	no	✓	–	–	–	–	–	–
first_perc_for_cutoff_decr	$\mathbb{Q}$	-0.02	✓	–	–	–	–	–	–
max_consecutive_infeasible	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_infeasibles	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_node	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_root	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
second_perc_for_cutoff_decr	$\mathbb{Q}$	-0.05	✓	–	–	–	–	–	–
Outer Approximation Decomposition (B-OA)									
oa_decomposition	string	no	–	–	✓	✓	✓	–	–
Outer Approximation cuts generation									
add_only_violated_oa	string	no	–	✓	✓	✓	✓	✓	✓
oa_cuts_scope	string	global	–	✓	✓	✓	✓	✓	✓
tiny_element	$\mathbb{Q}$	10^{-8}	–	✓	✓	✓	✓	✓	✓
very_tiny_element	$\mathbb{Q}$	10^{-17}	–	✓	✓	✓	✓	✓	✓
Output									
bb_log_interval	$\mathbb{Z}$	100	✓	✓	✓	✓	✓	✓	✓
bb_log_level	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	✓
fp_log_frequency	$\mathbb{Q}$	100	–	–	✓	✓	–	–	–
fp_log_level	$\mathbb{Z}$	1	–	–	✓	✓	–	–	–
lp_log_level	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
milp_log_level	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
nlp_log_at_root	$\mathbb{Z}$	5	✓	✓	✓	✓	✓	✓	–
nlp_log_level	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	✓
oa_cuts_log_level	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
oa_log_frequency	$\mathbb{Q}$	100	✓	–	–	✓	✓	–	–
oa_log_level	$\mathbb{Z}$	1	✓	–	–	✓	✓	–	–
Strong branching setup									
candidate_sort_criterion	string	best-ps-cost	✓	✓	✓	✓	✓	✓	–
maxmin_crit_have_sol	$\mathbb{Q}$	0.1	✓	✓	✓	✓	✓	✓	–

continued on next page

Option	type	default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Par
maxmin_crit_no_sol	$\mathbb{Q}$	0.7	✓	✓	✓	✓	✓	✓	–
min_number_strong_branch	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	–
number_before_trust_list	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	–
number_look_ahead	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	–	–
number_strong_branch_root	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	–
setup_pseudo_frac	$\mathbb{Q}$	0.5	✓	✓	✓	✓	✓	✓	–
trust_strong_branching_for_pseudo_cost	string	yes	✓	✓	✓	✓	✓	✓	–

In the following we give a detailed list of Bonmin options. The value on the right denotes the default value.

Algorithm choice

`algorithm` (`B-BB, B-OA, B-QG, B-Hyb, B-Ecp, B-iFP`) `B-BB`
Choice of the algorithm.
This will preset some of the options of bonmin depending on the algorithm choice.

`B-BB` simple branch-and-bound algorithm,

`B-OA` OA Decomposition algorithm,

`B-QG` Quesada and Grossmann branch-and-cut algorithm,

`B-Hyb` hybrid outer approximation based branch-and-cut,

`B-Ecp` ecp cuts based branch-and-cut a la FilMINT.

`B-iFP` Iterated Feasibility Pump for MINLP.

Branch-and-bound options

`allowable_fraction_gap` (real) 0.1
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.

`allowable_gap` (real) 0
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.

`cutoff` ($-10^{100} \leq$ real $\leq 10^{100}$) 10^{100}
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.

`cutoff_decr` ($-10^{10} \leq$ real $\leq 10^{10}$) 10^{-5}
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).

`enable_dynamic_nlp` (`no, yes`) `no`
Enable dynamic linear and quadratic rows addition in nlp

`integer_tolerance` ($0 <$ real) 10^{-6}
Set integer tolerance.
Any number within that value of an integer is considered integer.

`iteration_limit` ($0 \leq$ integer) ∞
Set the cumulated maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound.
value 0 deactivates option.

`nlp_failure_behavior` (`stop, fathom`) `stop`
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

whithin the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal.

`stop` Stop when failure happens.

`fathom` Continue when failure happens.

`node_comparison` (`best-bound, depth-first, breadth-first, dynamic, best-guess`) `best-bound`
Choose the node selection strategy.
Choose the strategy for selecting the next node to be processed.

`best-bound` choose node with the smallest bound,

`depth-first` Perform depth first search,

`breadth-first` Perform breadth 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).

`best-guess` choose node with smallest guessed integer solution

`node_limit` ($0 \leq$ integer) ∞
Set the maximum number of nodes explored in the branch-and-bound search.

`num_cut_passes` ($0 \leq$ integer) 1
Set the maximum number of cut passes at regular nodes of the branch-and-cut.

`num_cut_passes_at_root` ($0 \leq$ integer) 20
Set the maximum number of cut passes at regular nodes of the branch-and-cut.

`number_before_trust` ($0 \leq$ integer) 8
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.

`number_strong_branch` ($0 \leq$ integer) 20
Choose the maximum number of variables considered for strong branching.
Set the number of variables on which to do strong branching.

`solution_limit` ($0 \leq$ integer) ∞
Abort after that much integer feasible solution have been found by algorithm
value 0 deactivates option

`time_limit` ($0 \leq$ real) 1000
Set the global maximum computation time (in secs) for the algorithm.

`tree_search_strategy` (`top-node, dive, probed-dive, dfs-dive, dfs-dive-dynamic`) `probed-dive`
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.

`top-node` Always pick the top node as sorted by the node comparison function

`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 childs before continuing the dive at each level.

`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. Once a prescribed limit of backtracks is attained pick top node as sorted by the tree comparison function

`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.

`variable_selection` (`most-fractional, strong-branching, reliability-branching, curvature-estimator, qp-strong-branching, lp-strong-branching, nlp-strong-branching, osi-simple, osi-strong, random`)
`strong-branching`
Chooses variable selection strategy

`most-fractional` Choose most fractional variable `strong-branching` Perform strong branching

`reliability-branching` Use reliability branching

`curvature-estimator` Use curvature estimation to select branching variable

`qp-strong-branching` Perform strong branching with QP approximation

`lp-strong-branching` Perform strong branching with LP approximation

`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

`random` Method to choose branching variable randomly

ECP cuts generation

`ecp_abs_tol` ($0 \leq$ real) 10^{-6}
Set the absolute termination tolerance for ECP rounds.

`ecp_max_rounds` ($0 \leq$ integer) 5
Set the maximal number of rounds of ECP cuts.

`ecp_probability_factor` (real) 10
Factor appearing in formula for skipping ECP cuts.
Choosing -1 disables the skipping.

`ecp_rel_tol` ($0 \leq$ real) 0
Set the relative termination tolerance for ECP rounds.

`filmint_ecp_cuts` ($0 \leq$ integer) 0
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.

Feasibility checker using OA cuts

`feas_check_cut_types` (`outer-approx, Benders`) `outer-approx`
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

`outer-approx` Generate a set of Outer Approximations cuts.

`Benders` Generate a single Benders cut.

`feas_check_discard_policy` (`detect-cycles, keep-all, treated-as-normal`) `detect-cycles`
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 (usualy 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

`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)

`generate_benders_after_so_many_oa` ($0 \leq$ integer) 5000
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.

MILP Solver

`cpx_parallel_strategy` ($-1 \leq$ integer ≤ 1) 0
Strategy of parallel search mode in CPLEX.
-1 = opportunistic, 0 = automatic, 1 = deterministic (refer to CPLEX documentation)

`milp_solver` (`Cbc_D`, `Cbc_Par`, `Cplex`) `Cbc_D`
Choose the subsolver to solve MILP sub-problems in OA decompositions.
To use Cplex, a valid license is required.

- `Cbc_D` Coin Branch and Cut with its default
- `Cbc_Par` Coin Branch and Cut with passed parameters
- `Cplex` IBM CPLEX

`milp_strategy` (`find_good_sol`, `solve_to_optimality`) `find_good_sol`
Choose a strategy for MILPs.

- `find_good_sol` Stop sub milps when a solution improving the incumbent is found
- `solve_to_optimality` Solve MILPs to optimality

`number_cpx_threads` ($0 \leq$ integer) 0
Set number of threads to use with cplex.
(refer to CPLEX documentation)

MILP cutting planes in hybrid algorithm (B-Hyb)

`2mir_cuts` ($-100 \leq$ integer) 0
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.

`Gomory_cuts` ($-100 \leq$ integer) -5
Frequency k (in terms of nodes) for generating Gomory cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`clique_cuts` ($-100 \leq$ integer) -5
Frequency (in terms of nodes) for generating clique cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`cover_cuts` ($-100 \leq$ integer) 0
Frequency (in terms of nodes) for generating cover cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`flow_cover_cuts` ($-100 \leq$ integer) -5
Frequency (in terms of nodes) for generating flow cover cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`lift_and_project_cuts` ($-100 \leq$ integer) 0
Frequency (in terms of nodes) for generating lift-and-project cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`mir_cuts` ($-100 \leq$ integer) -5
Frequency (in terms of nodes) for generating MIR cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

`reduce_and_split_cuts` ($-100 \leq$ integer) 0
Frequency (in terms of nodes) for generating reduce-and-split cuts in branch-and-cut
See option `2mir_cuts` for the meaning of k.

MINLP Heuristics

`feasibility_pump_objective_norm` ($1 \leq$ integer ≤ 2) 1
Norm of feasibility pump objective function

`fp_pass_infeasible` (`no`, `yes`) `no`
Say whether feasibility pump should claim to converge or not

- `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.

`heuristic_RINS` (`no, yes`) `no`
if yes runs the RINS heuristic

`heuristic_dive_MIP_fractional` (`no, yes`) `no`
if yes runs the Dive MIP Fractional heuristic

`heuristic_dive_MIP_vectorLength` (`no, yes`) `no`
if yes runs the Dive MIP VectorLength heuristic

`heuristic_dive_fractional` (`no, yes`) `no`
if yes runs the Dive Fractional heuristic

`heuristic_dive_vectorLength` (`no, yes`) `no`
if yes runs the Dive VectorLength heuristic

`heuristic_feasibility_pump` (`no, yes`) `no`
whether the heuristic feasibility pump should be used

`pump_for_minlp` (`no, yes`) `no`
if yes runs FP for MINLP

NLP interface

`warm_start` (`none, optimum, interior_point`) `none`
Select the warm start method
This will affect the function getWarmStart(), and as a consequence the warm starting in the various algorithms.

`none` No warm start

`optimum` Warm start with direct parent optimum

`interior_point` Warm start with an interior point of direct parent

NLP solution robustness

`max_consecutive_failures` ($0 \leq$ integer) 10
(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).

`max_random_point_radius` ($0 <$ real) 100000
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)

`num_iterations_suspect` ($-1 \leq$ integer) -1
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 outputed in a file (set to -1 to deactivate this).

`num_retry_unsolved_random_point` ($0 \leq$ integer) 0
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.

`random_point_perturbation_interval` ($0 <$ real) 1
Amount by which starting point is perturbed when choosing to pick random point by perturbating starting point

`random_point_type` (`Jon, Andreas, Claudia`) `Jon`
method to choose a random starting point

`Jon` Choose random point uniformly between the bounds

`Andreas` perturb the starting point of the problem within a prescribed interval

`Claudia` perturb the starting point using the perturbation radius suffix information

NLP solves in hybrid algorithm (B-Hyb)

`nlp_solve_frequency` ($0 \leq$ integer) 10
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.

`nlp_solve_max_depth` ($0 \leq$ integer) 10
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.

`nlp_solves_per_depth` ($0 \leq$ real) 10^{100}
Set average number of nodes in the tree at which NLP relaxations are solved in B-Hyb for each depth.

Nonconvex problems

`coeff_var_threshold` ($0 \leq$ real) 0.1
Coefficient of variation threshold (for dynamic definition of cutoff_decr).

`dynamic_def_cutoff_decr` (`no, yes`) `no`
Do you want to define the parameter cutoff_decr dynamically?

`first_perc_for_cutoff_decr` (real) -0.02
The percentage used when, the coeff of variance is smaller than the threshold, to compute the cutoff_decr dynamically.

`max_consecutive_infeasible` ($0 \leq$ integer) 0
Number of consecutive infeasible subproblems before aborting a branch.
Will continue exploring a branch of the tree until "max_consecutive_infeasible" consecutive problems are infeasibles by the NLP sub-solver.

`num_resolve_at_infeasibles` ($0 \leq$ integer) 0
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.

`num_resolve_at_node` ($0 \leq$ integer) 0
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.

`num_resolve_at_root` ($0 \leq$ integer) 0
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.

`second_perc_for_cutoff_decr` (real) -0.05
The percentage used when, the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically.

Outer Approximation Decomposition (B-OA)

`oa_decomposition` (`no, yes`) `no`
If yes do initial OA decomposition

Outer Approximation cuts generation

`add_only_violated_oa` (`no, yes`) `no`
Do we add all OA cuts or only the ones violated by current point?

`no` Add all cuts

`yes` Add only violated Cuts

`oa_cuts_scope` (`local, global`) `global`
Specify if OA cuts added are to be set globally or locally valid

`local` Cuts are treated as locally valid `global` Cuts are treated as globally valid

`tiny_element` ($-0 \leq$ real) 10^{-8}
Value for tiny element in OA cut
We will remove "cleanly" (by relaxing cut) an element lower than this.

`very_tiny_element` ($-0 \leq$ real) 10^{-17}
Value for very tiny element in OA cut
Algorithm will take the risk of neglecting an element lower than this.

Output

`bb_log_interval` ($0 \leq$ integer) 100
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.

`bb_log_level` ($0 \leq$ integer ≤ 5) 1
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

`fp_log_frequency` ($0 <$ real) 100
display an update on lower and upper bounds in FP every n seconds

`fp_log_level` ($0 \leq$ integer ≤ 2) 1
specify FP iterations log level.
Set the level of output of OA decomposition solver : 0 - none, 1 - normal, 2 - verbose

`lp_log_level` ($0 \leq$ integer ≤ 4) 0
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

`milp_log_level` ($0 \leq$ integer ≤ 4) 0
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

`nlp_log_at_root` ($0 \leq$ integer ≤ 12) 5
Specify a different log level for root relaxtion.

`nlp_log_level` ($0 \leq$ integer ≤ 2) 1
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

`oa_cuts_log_level` ($0 \leq$ integer) 0
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.

`oa_log_frequency` ($0 <$ real) 100
display an update on lower and upper bounds in OA every n seconds

`oa_log_level` ($0 \leq$ integer ≤ 2) 1
specify OA iterations log level.
Set the level of output of OA decomposition solver : 0 - none, 1 - normal, 2 - verbose

Strong branching setup

`candidate_sort_criterion` (`best-ps-cost, worst-ps-cost, most-fractional, least-fractional`) `best-ps-cost`
Choice of the criterion to choose candidates in strong-branching

`best-ps-cost` Sort by decreasing pseudo-cost

`worst-ps-cost` Sort by increasing pseudo-cost `most-fractional` Sort by decreasing integer infeasibility

`least-fractional` Sort by increasing integer infeasibility

`maxmin_crit_have_sol` $(0 \leq \text{real} \leq 1)$ 0.1
Weight towards minimum in of lower and upper branching estimates when a solution has been found.

`maxmin_crit_no_sol` $(0 \leq \text{real} \leq 1)$ 0.7
Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.

`min_number_strong_branch` $(0 \leq \text{integer})$ 0
Sets minimum number of variables for strong branching (overriding trust)

`number_before_trust_list` $(-1 \leq \text{integer})$ 0
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"

`number_look_ahead` $(0 \leq \text{integer})$ 0
Sets limit of look-ahead strong-branching trials

`number_strong_branch_root` $(0 \leq \text{integer})$ ∞
Maximum number of variables considered for strong branching in root node.

`setup_pseudo_frac` $(0 \leq \text{real} \leq 1)$ 0.5
Proportion of strong branching list that has to be taken from most-integer-infeasible list.

`trust_strong_branching_for_pseudo_cost` (`no`, `yes`) `yes`
Whether or not to trust strong branching results for updating pseudo costs.

3 CBC

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: `https://projects.coin-or.org/Cbc`, `https://projects.coin-or.org/Cgl`, `https://projects.coin-or.org/Clp`. Most of the CBC documentation in the section was copied from the help in the CBC standalone version.

3.1 Model requirements

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).

3.2 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.

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.

Specification of Options

The GAMS/CBC 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 `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, available at `http://www.gams.com/download/download_old.htm`.

3.3 Options

Among all CBC options, the following GAMS parameters are currently supported in CBC: reslim, iterlim, nodlim, optca, optcr, cheat, cutoff, threads (only on non-Windows systems).

In the following, we summarize all available CBC options.

General Options

iterlim	iteration limit
names	specifies whether variable and equation names should be given to CBC
reslim	resource limit (CPU time in seconds)
special	options passed unseen to CBC
writemps	create MPS file for problem

LP Options

idiotcrash	idiot crash
sprintcrash	sprint crash
sifting	synonym for sprint crash
crash	use crash method to get dual feasible
maxfactor	maximum number of iterations between refactorizations
crossover	crossover to simplex algorithm after barrier
dualpivot	dual pivot choice algorithm
primalpivot	primal pivot choice algorithm
perturbation	perturbation of problem
scaling	scaling method
presolve	switch for initial presolve of LP
tol_dual	dual feasibility tolerance
tol_primal	primal feasibility tolerance
tol_presolve	tolerance used in presolve
passpresolve	how many passes to do in presolve
startalg	LP solver for root node

MIP Options

mipstart	whether it should be tried to use the initial variable levels as initial MIP solution
strategy	switches on groups of features
tol_integer	tolerance for integrality
sollim	limit on number of solutions
strongbranching	strong branching
trustpseudocosts	after howmany nodes we trust the pseudo costs
coststrategy	how to use costs as priorities
nodestrategy	how to select nodes
preprocess	integer presolve
threads	number of threads to use (available on Unix variants only)
printfrequency	frequency of status prints
increment	increment of cutoff when new incumbent
nodelim	node limit
nodlim	node limit
optca	absolute stopping tolerance
optcr	relative stopping tolerance
cutoff	cutoff for objective function value

MIP Options for Cutting Plane Generators

cutdepth	depth in tree at which cuts are applied
cut_passes_root	number of cut passes at root node
cut_passes_tree	number of cut passes at nodes in the tree
cuts	global switch for cutgenerators
cliquecuts	Clique Cuts
flowcovercuts	Flow Cover Cuts
gomorycuts	Gomory Cuts
knapsackcuts	Knapsack Cover Cuts
liftandprojectcuts	Lift and Project Cuts
mircuts	Mixed Integer Rounding Cuts
twomircuts	Two Phase Mixed Integer Rounding Cuts
probingcuts	Probing Cuts
reduceandsplitcuts	Reduce and Split Cuts
residualcapacitycuts	Residual Capacity Cuts

MIP Options for Heuristics

heuristics	global switch for heuristics
combinesolutions	combine solutions heuristic
dins	distance induced neighborhood search
divingrandom	turns on random diving heuristic
divingcoefficient	coefficient diving heuristic
divingfractional	fractional diving heuristic
divingguided	guided diving heuristic
divinglinesearch	line search diving heuristic
divingpseudocost	pseudo cost diving heuristic
divingvectorlength	vector length diving heuristic
feaspump	feasibility pump
feaspump_passes	number of feasibility passes
greedyheuristic	greedy heuristic

localtreesearch	local tree search heuristic
naiveheuristics	naive heuristics
pivotandfix	pivot and fix heuristic
randomizedrounding	randomized rounding heuristis
rens	relaxation enforced neighborhood search
rins	relaxed induced neighborhood search
roundingheuristic	rounding heuristic
vubheuristic	VUB heuristic

In the following, we give a detailed description of all available CBC options.

General Options

iterlim (*integer*)

For an LP, this is the maximum number of iterations to solve the LP. For a MIP, this option is ignored.

(default = GAMS iterlim)

names (*integer*)

This option causes GAMS names for the variables and equations to be loaded into Cbc. These names will then be used for error messages, log entries, and so forth. Turning names off may help if memory is very tight.

(default = 0)

0 Do not load variable and equation names.

1 Load variable and equation names.

reslim (*real*)

Maximum CPU time in seconds.

(default = GAMS reslim)

writemps (*string*)

Write the problem formulation in MPS format. The parameter value is the name of the MPS file.

special (*string*)

This parameter let you specify CBC options which are not supported by the GAMS/CoinCBC 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.

LP Options

idiotcrash (*integer*)

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.

sprintcrash ***(integer)***

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.

(default = -1)

-1	Let CLP decide by itself whether to use it.
0	Switch this method off.

sifting ***(integer)***

Synonym for sprintcrash.

(default = -1)

crash ***(string)***

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.

maxfactor ***(integer)***

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)

crossover ***(integer)***

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.

dualpivot ***(string)***

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.

primalpivot ***(string)***

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.

perturbation (*integer*)

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.

scaling (*string*)

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.

presolve (*integer*)

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.

tol_dual (*real*)

The maximum amount the dual constraints can be violated and still be considered feasible.

(default = 1e-7)

tol_primal (*real*)

The maximum amount the primal constraints can be violated and still be considered feasible.

(default = 1e-7)

tol_presolve (*real*)

The tolerance used in presolve.

(default = 1e-8)

passpresolve *(integer)*

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)

startalg *(string)*

Determines the algorithm to use for an LP or the initial LP relaxation if the problem is a MIP.

(default = dual)

primal	Let CLP use the primal simplex algorithm.
dual	Let CLP use the dual simplex algorithm.
barrier	Let CLP use a primal dual predictor corrector algorithm.

MIP Options

mipstart *(integer)*

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.

strategy *(integer)*

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.

(default = 1)

0	Use this setting for easy problems.
1	This is the default setting.
2	Use this setting for difficult problems.

tol_integer *(real)*

For an optimal solution, no integer variable may be farther than this from an integer value.

(default = 1e-6)

sollim *(integer)*

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.

strongbranching *(integer)*

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.

(default = 5)

trustpseudocosts (*integer*)

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)

coststrategy (*string*)

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.

nodestrategy (*string*)

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.

preprocess (*string*)

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.
equal	Turns on the presolve routines and let CBC turn inequalities with more than 5 elements into equalities (cliques) by adding slack variables.
equalall	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.
trysos	This option is similar to sos, but allows any number integer variables to be outside of the sets.

threads (*integer*)

This option controls the multithreading feature of CBC, which is currently available only on Unix variants.

(default = GAMS threads)

A number between 1 and 100 sets the number of threads used for parallel branch and bound. A number $100 + n$ with n between 1 and 100 says that n threads are used to parallelize the branch and bound, but also heuristics such as RINS which do branch and bound on a reduced model also use threads. A number $200+n$ with n between 1 and 100 says that n threads are used to parallelize the branch and bound, but also the cut generators at the root node (i.e., before threads are useful) are run in parallel. A number $300 + n$ with n between 1 and 100 combines the $100 + n$ and $200 + n$ options. A number $400 + n$ with n between 1 and 100 says that n threads are used in sub-trees. Thus, n threads are used to parallelize the branch and bound, but also heuristics use threads and the cut generators at the root node are run in parallel. The $100 + n$, $200 + n$, and $300 + n$ options are experimental.

printfrequency (*integer*)

Controls the number of nodes that are evaluated between status prints.

(default = 0)

0	Automatic choice, which is 100 for large problems and 1000 for small problems.

increment (*real*)

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)

nodelim (*integer*)

Maximum number of nodes that are considered in the Branch and Bound.

(default = GAMS nodlim)

nodlim (*integer*)

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 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 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)

cutoff (*real*)

CBC stops if the objective function values exceeds (in case of maximization) or falls below (in case of minimization) this value.

(default = GAMS cutoff)

MIP Options for Cutting Plane Generators

cutdepth (*integer*)

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.

cut_passes_root (*integer*)

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.

(default = 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 (*integer*)

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 = 1)

cuts (*string*)

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 behaviour).

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.

cliquecuts (*string*)

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 = ifmove)

flowcovercuts (*string*)

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)*

Determines whether and when CBC should try to generate mixed-integer Gomory cuts.

See the option cuts for an explanation on the different values.

Reference: Laurence A. Wolsey, Integer Programming, Wiley, John & Sons, (1998) 124-132.

(default = ifmove)

knapsackcuts *(string)*

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)*

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)

mircuts *(string)*

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)

twomircuts *(string)*

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. Guenluek, Valid Inequalities Based on Simple Mixed-integer Sets, to appear in Math. Programming.

(default = root)

probingcuts *(string)*

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 behaviour).
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.

reduceandsplitcuts (*string*)

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)

residualcapacitycuts (*string*)

Determines whether and when CBC should try to generate residual capacity cuts.

See the option cuts for an explanation on the different values.

These inequalities are particularly useful for Network Design and Capacity Planning models.

References:
T.L. Magnanti, P. Mirchandani, and R. Vachani, The convex hull of two core capacitated network design problems, Math. Programming, 60 (1993), pp. 233-250.
A. Atamturk and D. Rajan, On splittable and unsplittable flow capacitated network design arc-set polyhedra, Math. Programming, 92 (2002), pp. 315-333.

(default = off)

MIP Options for Heuristics

heuristics (*integer*)

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).

combinesolutions (*integer*)

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.

dins ***(integer)***

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.

divingrandom ***(integer)***

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.

divingcoefficient ***(integer)***

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)***

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)***

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)***

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)***

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.

divingvectorlength ***(integer)***

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.

feaspump (*integer*)

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*)

This fine tunes the feasibility pump heuristic by setting the number of passes.

(default = 20)

greedyheuristic (*string*)

This parameter control the use of a pair of greedy heuristic 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.

localtreesearch (*integer*)

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.

naiveheuristics (*integer*)

This parameter controls the use of some naive heuristics, e.g., fixing of all integers with costs to zero.¡BR>

(default = 0)

0 Turns the naive heuristics off.

1 Turns the naive heuristics on.

randomizedrounding (*integer*)

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.

rens (*integer*)

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.

pivotandfix (*integer*)

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.

rins (*integer*)

This parameter control 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*)

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.

vubheuristic (*integer*)

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

(default = -1)

4 Couenne

COUENNE (**C**onvex **O**ver and **U**nder **En**velopes for **N**onlinear **E**stimation) 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, and now at Lehigh University. The COIN-OR project leader for COUENNE is Pietro Belotti.

COUENNE solves convex and nonconvex MINLPs by an LP based spatial branch-and-bound algorithm that is similar to the algorithm used in BARON. 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.

For more information on the algorithm we refer to [4, 6] and the COUENNE web site `https://projects.coin-or.org/Couenne`. Most of the COUENNE documentation in this section is taken from the COUENNE manual [5].

4.1 Model requirements

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).

If GAMS/Couenne is called for a linear model, the interface directly calls CBC.

4.2 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 }
```

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.

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, please refer to the Sections 5.4 and 2.3 for a detailed description. The scheme to name option files is the same as for all other GAMS solvers. Specifying `optfile=1` let GAMS/Couenne read `couenne.opt`, `optfile=2` corresponds to `couenne.op2`, and so on. The format of the option file is the same as for Ipopt (see Section 5.2).

GAMS/Couenne understands currently the following GAMS parameters: `reslim` (time limit), `nodlim` (node limit), `cutoff`, `optca` (absolute gap tolerance), and `optcr` (relative gap tolerance). One can set them either on the command line, e.g. `nodlim=1000`, or inside your GAMS program, e.g. `Option nodlim=1000;`. Further, under Linux and Windows, the option `threads` can be used to control the number of threads used in the linear algebra routines of Ipopt.

4.3 Detailed Options Description

In the following we give a detailed list of options available for Couenne solely. The value on the right denotes the default value. Note that options specific to Ipopt and Bonmin are not listed her, see Sections 2.3 and 5.4 instead.

`2mir_cuts` ($-100 \leq$ integer) 0
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.

`Gomory_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating Gomory_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`aggressive_fbbt` (`no`, `yes`) `yes`
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.

`art_cutoff` (real) ∞
Artificial cutoff.

`art_lower` (real) $-\infty$
Artificial lower bound.

`boundtightening_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for bound tightening code in Couenne

`branch_conv_cuts` (`no`, `yes`) `yes`
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.

`branch_fbbt` (`no`, `yes`) `yes`
Apply bound tightening before branching
After applying a branching rule and before re-solving the subproblem, apply Bound Tightening.

`branch_lp_clamp` ($0 \leq$ real ≤ 1) 0.2
Defines safe interval percentage for using LP point as a branching point.

`branch_lp_clamp_cube` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_div` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_exp` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_log` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_negpow` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_pow` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_prod` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_sqr` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_lp_clamp_trig` ($0 \leq$ real ≤ 0.5) 0.2
Defines safe interval percentage [0,0.5] for using LP point as a branching point.

`branch_midpoint_alpha` ($0 \leq$ real ≤ 1) 0.25
Defines convex combination of mid point and current LP point: b = alpha x_lp + (1-alpha) (lb+ub)/2.

`branch_pt_select` (`lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `mid-point`
Chooses branching point selection strategy

- `lp-clamped` LP point clamped in [k,1-k] of the bound intervals (k defined by lp_clamp)
- `lp-central` LP point if within [k,1-k] of the bound intervals, middle point otherwise(k defined by branch_lp_clamp)
- `balanced` minimizes max distance from curve to convexification
- `min-area` minimizes total area of the two convexifications
- `mid-point` convex combination of current point and mid point
- `no-branch` do not branch, return null infeasibility; for testing purposes only

`branch_pt_select_cube` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`
Chooses branching point selection strategy for operator cube. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_div` (`common`, `lp-clamped`, `lp-central`, `balanced`, `min-area`, `mid-point`, `no-branch`) `common`

Chooses branching point selection strategy for operator div. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_exp` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator exp. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_log` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator log. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_negpow` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator negpow. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_pow` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator pow. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_prod` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator prod. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_sqr` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator sqr. Default is to use the value of `branch_pt_select` (value `common`).

`branch_pt_select_trig` (`common, lp-clamped, lp-central, balanced, min-area, mid-point, no-branch`) `common`
Chooses branching point selection strategy for operator trig. Default is to use the value of `branch_pt_select` (value `common`).

`branching_object` (`vt_obj, var_obj, expr_obj`) `var_obj`
type of branching object for variable selection

`vt_obj`	use Violation Transfer from Tawarmalani and Sahinidis
`var_obj`	use one object for each variable
`expr_obj`	use one object for each nonlinear expression

`branching_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for braching code in Couenne

`check_lp` (`no, yes`) `no`
Check all LPs through an independent call to OsiClpSolverInterface::initialSolve()

`clique_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating clique_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`cont_var_priority` ($1 \leq$ integer) 2000
Priority of continuous variable branching
When branching, this is compared to the priority of integer variables, whose priority is fixed to 1000, and SOS, whose priority is 10. Higher values mean smaller priority, so if this parameter is set to 1001 or higher, if a branch-and-bound node has at least one integer variable whose value is fractional, then branching will be performed on that variable.

`convexification_cuts` ($-99 \leq$ integer) 1
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.

`convexification_points` ($0 \leq$ integer) 4
Specify the number of points at which to convexify when convexification type is uniform-grid or around-current-point.

`convexification_type` (`current-point-only, uniform-grid, around-current-point`) `current-point-only`
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.

`current-point-only` Only at current optimum of relaxation

`uniform-grid` Points chosen in a uniform grid between the bounds of the problem

`around-current-point` At points around current optimum of relaxation

`convexifying_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for convexifying code in Couenne

`cover_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating cover_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`delete_redundant` (`no, yes`) `yes`
Eliminate redundant variables, which appear in the problem as x_k = x_h

`no` Keep redundant variables, making the problem a bit larger

`yes` Eliminate redundant variables (the problem will be equivalent, only smaller)

`disj_active_cols` (`yes, no`) `no`
Only include violated variable bounds in the Cut Generating LP (CGLP).
This reduces the size of the CGLP, but may produce less efficient cuts.

`disj_active_rows` (`yes, no`) `no`
Only include violated linear inequalities in the CGLP.
This reduces the size of the CGLP, but may produce less efficient cuts.

`disj_cumulative` (`yes, no`) `no`
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.

`disj_depth_level` ($-1 \leq$ integer) 5
Depth of the B&B tree when to start decreasing the number of objects that generate disjunctions.
This has a similar behavior as log_num_obbt_per_level. A value of -1 means that generation can be done at all nodes.

`disj_depth_stop` ($-1 \leq$ integer) 20
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

`disj_init_number` ($-1 \leq$ integer) 10
Maximum number of disjunction to consider at each iteration.
-1 means no limit.

`disj_init_perc` ($0 \leq$ real ≤ 1) 0.5
The maximum fraction of all disjunctions currently violated by the problem to consider for generating disjunctions.

`disjcuts_print_level` ($-2 \leq$ integer ≤ 12) 0
Output level for disjunctive cuts in Couenne

`display_stats` (`yes, no`) `no`
display statistics at the end of the run

`enable_lp_implied_bounds` (`no, yes`) `no`
Enable OsiSolverInterface::tightenBounds () – warning: it has caused some trouble to Couenne

`estimate_select` (`normal, product`) `normal`
How the min/max estimates of the subproblems' bounds are used in strong branching

- `normal` as usual in literature
- `product` use their product

`feas_tolerance` (real) 10^{-5}
Tolerance for constraints/auxiliary variables
Default value is zero.

`feasibility_bt` (`no, yes`) `yes`
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.

`flow_covers_cuts` ($-100 \leq$ integer) 0
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.

`lift_and_project_cuts` ($-100 \leq$ integer) 0
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.

`local_optimization_heuristic` (`no, yes`) `yes`
Do we search for local solutions of NLP's
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.

`log_num_abt_per_level` ($-1 \leq$ integer) 2
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 $2^{(k-level)}$, level being the current depth of the B&B tree.

`log_num_local_optimization_per_level` ($-1 \leq$ integer) 2
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.

`log_num_obbt_per_level` ($-1 \leq$ integer) 1
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 $2^{(k-level)}$, level being the current depth of the B&B tree.

`lp_solver` (`clp, cplex`) `clp`
Linear Programming solver for the linear relaxation.

- `clp` Use the Coin-OR Open Source solver CLP
- `cplex` Use the commercial solver Cplex (license is needed)

`minlp_disj_cuts` ($-99 \leq$ integer) 0
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.

`mir_cuts` ($-100 \leq$ integer) 0
Frequency k (in terms of nodes) for generating mir_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`nlpheur_print_level` $(-2 \leq \text{integer} \leq 12)$ 0
Output level for NLP heuristic in Couenne

`opt_window` (real) ∞
Window around known optimum.

`optimality_bt` `(no, yes)` `yes`
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.

`orbital_branching` `(yes, no)` `no`
detect symmetries and apply orbital branching

`probing_cuts` $(-100 \leq \text{integer})$ 0
Frequency k (in terms of nodes) for generating probing_cuts cuts in branch-and-cut.
See option `2mir_cuts` for the meaning of k.

`problem_print_level` $(-2 \leq \text{integer} \leq 12)$ 1
Output level for problem manipulation code in Couenne

`pseudocost_mult` `(infeasibility, projectDist, interval_lp, interval_lp_rev, interval_br, interval_br_rev)`
`interval_br_rev`
Multipliers of pseudocosts for estimating and update estimation of bound

- `infeasibility` infeasibility returned by object
- `projectDist` distance between current LP point and resulting branches' LP points
- `interval_lp` width of the interval between bound and current lp point
- `interval_lp_rev` similar to interval_lp, reversed
- `interval_br` width of the interval between bound and branching point
- `interval_br_rev` similar to interval_br, reversed

`pseudocost_mult_lp` `(yes, no)` `no`
Use distance between LP points to update multipliers of pseudocosts after simulating branching

`red_cost_branching` `(no, yes)` `no`
Apply Reduced Cost Branching (instead of the Violation Transfer) – MUST have vt_obj enabled

- `no` Use Violation Transfer with $\sum |\pi\_ia\_ij|$
- `yes` Use Reduced cost branching with $|\sum \pi\_ia\_ij|$

`redcost_bt` `(no, yes)` `yes`
Reduced cost bound tightening
This bound reduction technique uses the reduced costs of the LP in order to infer better variable bounds.

`reduce_split_cuts` $(-100 \leq \text{integer})$ 0
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.

`reformulate_print_level` $(-2 \leq \text{integer} \leq 12)$ 0
Output level for reformulating problems in Couenne

`use_quadratic` `(no, yes)` `no`
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.

- `no` Use an auxiliary for each bilinear term
- `yes` Create only one auxiliary for a quadratic expression

`violated_cuts_only` `(no, yes)` `yes`
Yes if only violated convexification cuts should be added

5 Ipopt

Ipopt (**I**nterior **P**oint **Opt**imizer) is an open-source solver for large-scale nonlinear programming. The code has been written primarily by Andreas Wächter, who is the COIN-OR project leader for Ipopt.

Ipopt implements an interior point line search filter method. For more information on the algorithm we refer to [22, 6] and the Ipopt web site https://projects.coin-or.org/Ipopt. Most of the Ipopt documentation in the section was taken from the Ipopt manual [18].

GAMS/Ipopt uses MUMPS 4.9 [2, 3] as linear solver, cf. http://graal.ens-lyon.fr/MUMPS.

5.1 Model requirements

Ipopt can handle nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable.

5.2 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 }
```

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.

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 4.9 [2, 3]. The user can provide the routines from the Harwell Subroutine Library (HSL) as shared (or dynamic) libraries to replace MUMPS.

Using Harwell Subroutine Library routines with GAMS/Ipopt. GAMS/Ipopt can use the HSL routines `MA27`, `MA28`, `MA57`, and `MC19` 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. You can also specify the path and name for this library with the option `hsl_library`. For example,

```
 linear_solver ma27
 hsl_library   /my/path/to/the/hsllib/myhsllib.so
```

tells Ipopt to use the linear solver MA27 from the HSL library `myhsllib.so` under the specified path.

The HSL routines `MA27`, `MA28`, and `MC19` are available at http://www.cse.clrc.ac.uk/nag/hsl. Note that it is your responsibility to ensure that you are entitled to download and use these routines! You can build a shared library using the ThirdParty/HSL project at COIN-OR.

Specification of Options

Ipopt has many options that can be adjusted for the algorithm (see Section 5.4). 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 `ipopt.opt` file in the directory you are executing Ipopt.

The `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. Don't forget to ensure you have a newline at the end of the file. For example,

```
# 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
```

is a valid `ipopt.opt` file.

GAMS/Ipopt understand currently the following GAMS parameters: `reslim` (time limit), `iterlim` (iteration limit), `domlim` (domain violation limit). You can set them either on the command line, e.g. `iterlim=500`, or inside your GAMS program, e.g. `Option iterlim=500;`. Further, under Linux and Windows, the option `threads` can be used to control the number of threads used in the basic linear algebra routines.

5.3 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,

```
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 2.05e+00    -  2.14e-01 1.00e+00f  1
   2  1.8719906e+01 1.25e-02 9.04e+00  -2.2 5.94e-02   2.0 8.04e-01 1.00e+00h  1
```

and the columns of output are defined as

iter 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 scaled primal infeasibility at the current point. 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. 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.

lg(rg) $\log_{10}$ of the value of the regularization term for the Hessian of the Lagrangian in the augmented system.

alpha_du The stepsize for the dual variables.

alpha_pr The stepsize for the primal variables.

ls The number of backtracking line search steps.

When the algorithm terminates, Ipopt will output a message to the screen based on the return status of the call to Optimize. The following is a list of the possible output messages to the console, 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.

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 ℓ_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 divering; 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 time limit 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 option max_iter.

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 messages 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 `https://projects.coin-or.org/GAMSlinks` or `https://projects.coin-or.org/Ipopt`).

5.4 Detailed Options Description

Barrier Parameter Update

`adaptive_mu_globalization` (`kkt-error, obj-constr-filter, never-monotone-mode`) `obj-constr-filter`
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".)

- `kkt-error` nonmonotone decrease of kkt-error
- `obj-constr-filter` 2-dim filter for objective and constraint violation
- `never-monotone-mode` disables globalization

`adaptive_mu_kkt_norm_type` (`1-norm, 2-norm-squared, max-norm, 2-norm`) `2-norm-squared`
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.

- `1-norm` use the 1-norm (abs sum)
- `2-norm-squared` use the 2-norm squared (sum of squares)
- `max-norm` use the infinity norm (max)
- `2-norm` use 2-norm

`adaptive_mu_kkterror_red_fact` ($0 < \text{real} < 1$) 0.9999
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.

`adaptive_mu_kkterror_red_iters` ($0 \leq \text{integer}$) 4
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.

`adaptive_mu_monotone_init_factor` ($0 < \text{real}$) 0.8
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".

`adaptive_mu_restore_previous_iterate` (`no, yes`) `no`
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).

- `no` don't restore accepted iterate
- `yes` restore accepted iterate

`barrier_tol_factor` ($0 < \text{real}$) 10
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).

`filter_margin_fact` ($0 < \text{real} < 1$) 10^{-5}
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".

`filter_max_margin` (0 < real) 1
Maximum width of margin in obj-constr-filter adaptive globalization strategy.

`fixed_mu_oracle (probing, loqo, quality-function, average_compl)` `average_compl`
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".)

`probing` Mehrotra's probing heuristic

`loqo` LOQO's centrality rule

`quality-function` minimize a quality function

`average_compl` base on current average complementarity

`mu_allow_fast_monotone_decrease (no, yes)` `yes`
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.

`no` Take at least one iteration per barrier problem

`yes` Allow fast decrease of mu if barrier test it met

`mu_init` (0 < real) 0.1
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")

`mu_linear_decrease_factor` (0 < real < 1) 0.2
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.

`mu_max` (0 < real) 100000
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".)

`mu_max_fact` (0 < real) 1000
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".)

`mu_min` (0 < real) 10^{-11}
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".)

`mu_oracle (probing, loqo, quality-function)` `quality-function`
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").

`probing` Mehrotra's probing heuristic

`loqo` LOQO's centrality rule

`quality-function` minimize a quality function

`mu_strategy` (`monotone, adaptive`) `adaptive`
Update strategy for barrier parameter.
Determines which barrier parameter update strategy is to be used.

- `monotone` use the monotone (Fiacco-McCormick) strategy
- `adaptive` use the adaptive update strategy

`mu_superlinear_decrease_power` ($1 <$ real < 2) 1.5
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$\hat{}$"superlinear_decrease_power". (This is theta_mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

`quality_function_balancing_term` (`none, cubic`) `none`
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".)

- `none` no balancing term is added
- `cubic` Max(0,Max(dual_inf,primal_inf)-compl)$\hat{3}$

`quality_function_centrality` (`none, log, reciprocal, cubed-reciprocal`) `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".)

- `none` no penalty term is added
- `log` complementarity * the log of the centrality measure
- `reciprocal` complementarity * the reciprocal of the centrality measure
- `cubed-reciprocal` complementarity * the reciprocal of the centrality measure cubed

`quality_function_max_section_steps` ($0 \leq$ integer) 8
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".)

`quality_function_norm_type` (`1-norm, 2-norm-squared, max-norm, 2-norm`) `2-norm-squared`
Norm used for components of the quality function.
(Only used if option "mu_oracle" is set to "quality-function".)

- `1-norm` use the 1-norm (abs sum)
- `2-norm-squared` use the 2-norm squared (sum of squares)
- `max-norm` use the infinity norm (max)
- `2-norm` use 2-norm

`quality_function_section_qf_tol` ($0 \leq$ real < 1) 0
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".)

`quality_function_section_sigma_tol` ($0 \leq$ real < 1) 0.01
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".)

`sigma_max` ($0 <$ real) 100
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_min` ($0 \leq$ real) 10^{-6}
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".)

`tau_min` ($0 <$ real < 1) 0.99
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.

Convergence

`acceptable_compl_inf_tol` ($0 <$ real) 0.01
"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.

`acceptable_constr_viol_tol` ($0 <$ real) 0.01
"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.

`acceptable_dual_inf_tol` ($0 <$ real) 10^{10}
"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.

`acceptable_iter` ($0 \leq$ integer) 15
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.

`acceptable_obj_change_tol` ($0 \leq$ real) 10^{20}
"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.

`acceptable_tol` ($0 <$ real) 10^{-6}
"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.

`compl_inf_tol` ($0 <$ real) 0.0001
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.

`constr_viol_tol` ($0 <$ real) 0.0001
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.

`diverging_iterates_tol` ($0 <$ real) 10^{20}
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.

`dual_inf_tol` ($0 <$ real) 1
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.

`max_cpu_time` ($0 <$ real) 1000
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_iter` ($0 \leq$ integer) ∞
Maximum number of iterations.
The algorithm terminates with an error message if the number of iterations exceeded this number.

`mu_target` ($0 \leq$ real) 0
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.

`s_max` ($0 <$ real) 100
Scaling threshold for the NLP error.
(See paragraph after Eqn. (6) in the implementation paper.)

`tol` ($0 <$ real) 10^{-8}
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", "primal_inf_tol", and "cmpl_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.

Hessian Approximation

`hessian_approximation (exact, limited-memory)` `exact`
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.

`exact` Use second derivatives provided by the NLP.

`limited-memory` Perform a limited-memory quasi-Newton approximation

`hessian_approximation_space (nonlinear-variables, all-variables)` `nonlinear-variables`
Indicates in which subspace the Hessian information is to be approximated.

`nonlinear-variables` only in space of nonlinear variables.

`all-variables` in space of all variables (without slacks)

`limited_memory_aug_solver (sherman-morrison, extended)` `sherman-morrison`
Strategy for solving the augmented system for low-rank Hessian.

`sherman-morrison` use Sherman-Morrison formula

`extended` use an extended augmented system

`limited_memory_init_val` ($0 <$ real) 1
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".

`limited_memory_init_val_max` ($0 <$ real) 10^8
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".

`limited_memory_init_val_min` ($0 <$ real) 10^{-8}
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".

`limited_memory_initialization` (`scalar1, scalar2, scalar3, scalar4, constant`) `scalar1`
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.

- `scalar1` sigma = $s^T y/s^T s$
- `scalar2` sigma = $y^T y/s^T y$
- `scalar3` arithmetic average of scalar1 and scalar2
- `scalar4` geometric average of scalar1 and scalar2
- `constant` sigma = limited_memory_init_val

`limited_memory_max_history` ($0 \leq$ integer) 6
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.

`limited_memory_max_skipping` ($1 \leq$ integer) 2
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.

`limited_memory_update_type` (`bfgs, sr1`) `bfgs`
Quasi-Newton update formula for the limited memory approximation.
Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

- `bfgs` BFGS update (with skipping)
- `sr1` SR1 (not working well)

Initialization

`bound_frac` ($0 <$ real ≤ 0.5) 0.01
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.)

`bound_mult_init_method` (`constant, mu-based`) `constant`
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.

- `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` ($0 <$ real) 1
Initial value for the bound multipliers.
All dual variables corresponding to bound constraints are initialized to this value.

`bound_push` ($0 <$ real) 0.01
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.)

`constr_mult_init_max` ($0 \leq$ real) 1000
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.

`least_square_init_duals` (`no, yes`) `no`
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".

`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` (`no, yes`) `no`
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.

`no` take user-provided point

`yes` overwrite user-provided point with least-square estimates

`slack_bound_frac` ($0 <$ real ≤ 0.5) 0.01
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.)

`slack_bound_push` ($0 <$ real) 0.01
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.)

Line Search

`accept_after_max_steps` ($-1 \leq$ integer) -1
Accept a trial point after maximal this number of steps.
Even if it does not satisfy line search conditions.

`accept_every_trial_step` (`no, yes`) `no`
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.

`no` don't arbitrarily accept the full step

`yes` always accept the full step

`alpha_for_y` (`primal, bound-mult, min, max, full, min-dual-infeas, safer-min-dual-infeas, primal-and-full`
`dual-and-full, acceptor`) `primal`
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.

`primal` use primal step size

`bound-mult` use step size for the bound multipliers (good for LPs)

`min` use the min of primal and bound multipliers `max` use the max of primal and bound multipliers

`full` take a full step of size one

`min-dual-infeas` choose step size minimizing new dual infeasibility

`safer-min-dual-infeas` like "min_dual_infeas", but safeguarded by "min" and "max"

`primal-and-full` use the primal step size, and full step if delta_x <= alpha_for_y_tol

`dual-and-full` use the dual step size, and full step if delta_x <= alpha_for_y_tol

`acceptor` Call LSAcceptor to get step size for y

`alpha_for_y_tol` ($0 \leq$ real) 10
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.

`alpha_min_frac` ($0 <$ real < 1) 0.05
Safety factor for the minimal step size (before switching to restoration phase).
(This is gamma_alpha in Eqn. (20) in the implementation paper.)

`alpha_red_factor` ($0 <$ real < 1) 0.5
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.

`constraint_violation_norm_type` (`1-norm, 2-norm, max-norm`) `1-norm`
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.

`1-norm` use the 1-norm

`2-norm` use the 2-norm

`max-norm` use the infinity norm

`corrector_compl_avrg_red_fact` ($0 <$ real) 1
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.

`corrector_type` (`none, affine, primal-dual`) `none`
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.

`none` no corrector

`affine` corrector step towards mu=0

`primal-dual` corrector step towards current mu

`delta` ($0 <$ real) 1
Multiplier for constraint violation in the switching rule.
(See Eqn. (19) in the implementation paper.)

`eta_phi` ($0 <$ real < 0.5) 10^{-8}
Relaxation factor in the Armijo condition.
(See Eqn. (20) in the implementation paper)

`filter_reset_trigger` ($1 \leq$ integer) 5
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.

`gamma_phi` ($0 <$ real < 1) 10^{-8}
Relaxation factor in the filter margin for the barrier function.
(See Eqn. (18a) in the implementation paper.)

`gamma_theta` ($0 <$ real < 1) 10^{-5}
Relaxation factor in the filter margin for the constraint violation.

(See Eqn. (18b) in the implementation paper.)

`kappa_sigma` ($0 <$ real) 10^{10}
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.

`kappa_soc` ($0 <$ real) 0.99
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.)

`max_filter_resets` ($0 \leq$ integer) 5
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.

`max_soc` ($0 \leq$ integer) 4
Maximum number of second order correction trial steps at each iteration.
Choosing 0 disables the second order corrections. (This is $p\hat{m}ax$ of Step A-5.9 of Algorithm A in the implementation paper.)

`nu_inc` ($0 <$ real) 0.0001
Increment of the penalty parameter.

`nu_init` ($0 <$ real) 10^{-6}
Initial value of the penalty parameter.

`obj_max_inc` ($1 <$ real) 5
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.

`recalc_y (no, yes)` `no`
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.

`no` use the Newton step to update the multipliers

`yes` use least-square multiplier estimates

`recalc_y_feas_tol` ($0 <$ real) 10^{-6}
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.

`rho` ($0 <$ real < 1) 0.1
Value in penalty parameter update formula.

`s_phi` ($1 <$ real) 2.3
Exponent for linear barrier function model in the switching rule.
(See Eqn. (19) in the implementation paper.)

`s_theta` ($1 <$ real) 1.1
Exponent for current constraint violation in the switching rule.
(See Eqn. (19) in the implementation paper.)

`skip_corr_if_neg_curv (no, yes)` `yes`
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".

`no` don't skip

`yes` skip

`skip_corr_in_monotone_mode` (`no`, `yes`) `yes`
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".

`no` don't skip

`yes` skip

`slack_move` ($0 \leq$ real) $1.81899 \cdot 10^{-12}$
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$\hat{3}/4$. (See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.)

`theta_max_fact` ($0 <$ real) 10000
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).

`theta_min_fact` ($0 <$ real) 0.0001
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).

`tiny_step_tol` ($0 \leq$ real) $2.22045 \cdot 10^{-15}$
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.

`tiny_step_y_tol` ($0 \leq$ real) 0.01
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.

`watchdog_shortened_iter_trigger` ($0 \leq$ integer) 10
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.

`watchdog_trial_iter_max` ($1 \leq$ integer) 3
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.

Linear Solver

`hsl_library` (`string`)
path and filename of HSL library for dynamic load
Specify the path to a library that contains HSL routines and can be load via dynamic linking. Note, that you still need to specify to use the corresponding routines (ma27, ...) by setting the corresponding options, e.g., "linear_solver".

`linear_scaling_on_demand` (`no`, `yes`) `yes`
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.

`no` Always scale the linear system.

`yes` Start using linear system scaling if solutions seem not good.

`linear_solver (ma27, ma57, mumps)` `ma27`
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 in order to use MA27 or MA57, a library with HSL code need to be provided (see Section 5.2 and the option "hsl_library").

`ma27` use the Harwell routine MA27

`ma57` use the Harwell routine MA57

`mumps` use MUMPS package

`linear_system_scaling (none, mc19, slack-based)` `mc19`
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.

`none` no scaling will be performed

`mc19` use the Harwell routine MC19

`slack-based` use the slack values

MA27 Linear Solver

`ma27_ignore_singularity (no, yes)` `no`
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.

`no` Don't have MA27 solve singular systems

`yes` Have MA27 solve singular systems

`ma27_la_init_factor` ($1 \leq$ real) 5
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.

`ma27_liw_init_factor` ($1 \leq$ real) 5
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.

`ma27_meminc_factor` ($1 \leq$ real) 10
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.

`ma27_pivtol` ($0 <$ real < 1) 10^{-8}
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.

`ma27_pivtolmax` ($0 <$ real < 1) 0.0001
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.

`ma27_skip_inertia_check` (`no, yes`) `no`
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.

`no` check inertia

`yes` skip inertia check

MA28 Linear Solver

`ma28_pivtol` ($0 < \text{real} \leq 1$) 0.01
Pivot tolerance for linear solver MA28.
This is used when MA28 tries to find the dependent constraints.

MA57 Linear Solver

`ma57_automatic_scaling` (`no, yes`) `yes`
Controls MA57 automatic scaling
This option controls the internal scaling option of MA57.This is ICNTL(15) in MA57.

`no` Do not scale the linear system matrix

`yes` Scale the linear system matrix

`ma57_block_size` ($1 \leq \text{integer}$) 16
Controls block size used by Level 3 BLAS in MA57BD
This is ICNTL(11) in MA57.

`ma57_node_amalgamation` ($1 \leq \text{integer}$) 16
Node amalgamation parameter
This is ICNTL(12) in MA57.

`ma57_pivot_order` ($0 \leq \text{integer} \leq 5$) 5
Controls pivot order in MA57
This is ICNTL(6) in MA57.

`ma57_pivtol` ($0 < \text{real} < 1$) 10^{-8}
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.

`ma57_pivtolmax` ($0 < \text{real} < 1$) 0.0001
Maximum pivot tolerance for the linear solver MA57.
Ipopt may increase pivtol as high as ma57_pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.

`ma57_pre_alloc` ($1 \leq \text{real}$) 1.05
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.

`ma57_small_pivot_flag` ($0 \leq \text{integer} \leq 1$) 0
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.

Mumps Linear Solver

`mumps_dep_tol` (real) -1
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.

`mumps_mem_percent` ($0 \leq$ integer) 1000
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.

`mumps_permuting_scaling` ($0 \leq$ integer ≤ 7) 7
Controls permuting and scaling in MUMPS
This is ICNTL(6) in MUMPS.

`mumps_pivot_order` ($0 \leq$ integer ≤ 7) 7
Controls pivot order in MUMPS
This is ICNTL(7) in MUMPS.

`mumps_pivtol` ($0 \leq$ real ≤ 1) 10^{-6}
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.

`mumps_pivtolmax` ($0 \leq$ real ≤ 1) 0.1
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.

`mumps_scaling` ($-2 \leq$ integer ≤ 77) 77
Controls scaling in MUMPS
This is ICNTL(8) in MUMPS.

NLP

`bound_relax_factor` ($0 \leq$ real) 10^{-10}
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.)

`check_derivatives_for_naninf (no, yes)` `no`
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!

`no` Don't check (faster).

`yes` Check Jacobians and Hessian for Nan and Inf.

`dependency_detection_with_rhs (no, yes)` `no`
Indicates if the right hand sides of the constraints should be considered during dependency detection

`no` only look at gradients

`yes` also consider right hand side

`dependency_detector (none, mumps, wsmp, ma28)` `none`
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.

`none` don't check; no extra work at beginning

`mumps` use MUMPS

`wsmp` use WSMP

`ma28` use MA28

`fixed_variable_treatment` (`make_parameter, make_constraint, relax_bounds`) `make_parameter`
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.

`make_parameter` Remove fixed variable from optimization variables

`make_constraint` Add equality constraints fixing variables

`relax_bounds` Relax fixing bound constraints

`honor_original_bounds` (`no, yes`) `yes`
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.

`no` Leave final point unchanged

`yes` Project final point back into original bounds

`jac_c_constant` (`no, yes`) `no`
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.

`no` Don't assume that all equality constraints are linear

`yes` Assume that equality constraints Jacobian are constant

`jac_d_constant` (`no, yes`) `no`
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.

`no` Don't assume that all inequality constraints are linear

`yes` Assume that equality constraints Jacobian are constant

`kappa_d` ($0 \leq$ real) 10^{-5}
Weight for linear damping term (to handle one-sided bounds).
(see Section 3.7 in implementation paper.)

`num_linear_variables` ($0 \leq$ integer) 0
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.

NLP Scaling

`nlp_scaling_constr_target_gradient` ($0 \leq$ real) 0
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.

`nlp_scaling_max_gradient` ($0 <$ real) 100
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".

`nlp_scaling_method` (`none, user-scaling, gradient-based, equilibration-based`) `gradient-based`
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")

`none` no problem scaling will be performed

`user-scaling` scaling parameters will come from the user

`gradient-based` scale the problem so the maximum gradient at the starting point is scaling_max_gradient

`equilibration-based` scale the problem so that first derivatives are of order 1 at random points (only available with MC19)

`nlp_scaling_min_value` ($0 \le$ real) 10^{-8}
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".

`nlp_scaling_obj_target_gradient` ($0 \le$ real) 0
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.

Output

`print_eval_error` (`no, yes`) `no`
whether to print information about function evaluation errors into the listing file

`print_info_string` (`no, yes`) `no`
Enables printing of additional info string at end of iteration output.
This string contains some insider information about the current iteration.

`no` don't print string

`yes` print string at end of each iteration output

`print_level` ($0 \le$ integer ≤ 12) 5
Output verbosity level.
Sets the default verbosity level for console output. The larger this value the more detailed is the output.

`print_timing_statistics` (`no, yes`) `no`
Switch to print timing statistics.
If selected, the program will print the CPU usage (user time) for selected tasks.

`no` don't print statistics

`yes` print all timing statistics

`replace_bounds` (`no, yes`) `no`
Indicates if all variable bounds should be replaced by inequality constraints
This option must be set for the inexact algorithm

`no` leave bounds on variables

`yes` replace variable bounds by inequality constraints

Restoration Phase

`bound_mult_reset_threshold` ($0 \le$ real) 1000
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.

`constr_mult_reset_threshold` ($0 \leq$ real) 0
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.

`evaluate_orig_obj_at_resto_trial` (`no, yes`) `yes`
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.

`no` skip evaluation

`yes` evaluate at every trial point

`expect_infeasible_problem` (`no, yes`) `no`
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.

`no` the problem probably be feasible

`yes` the problem has a good chance to be infeasible

`expect_infeasible_problem_ctol` ($0 \leq$ real) 0.001
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` ($0 <$ real) 10^8
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.

`max_resto_iter` ($0 \leq$ integer) 3000000
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.

`max_soft_resto_iters` ($0 \leq$ integer) 10
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.

`required_infeasibility_reduction` ($0 \leq$ real < 1) 0.9
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.

`resto_penalty_parameter` ($0 <$ real) 1000
Penalty parameter in the restoration phase objective function.
This is the parameter rho in equation (31a) in the Ipopt implementation paper.

`soft_resto_pderror_reduction_factor` ($0 \leq$ real) 0.9999
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.

`start_with_resto` (`no, yes`) `no`
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.

`no` don't force start in restoration phase

`yes` force start in restoration phase

Step Calculation

`fast_step_computation` (`no, yes`) `no`
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.

`no` Verify solution of linear system by computing residuals.

`yes` Trust that linear systems are solved well.

`first_hessian_perturbation` ($0 <$ real) 0.0001
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.)

`jacobian_regularization_exponent` ($0 \leq$ real) 0.25
Exponent for mu in the regularization for rank-deficient constraint Jacobians.
(This is kappa_c in the implementation paper.)

`jacobian_regularization_value` ($0 \leq$ real) 10^{-8}
Size of the regularization for rank-deficient constraint Jacobians.
(This is bar delta_c in the implementation paper.)

`max_hessian_perturbation` ($0 <$ real) 10^{20}
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$\hat{m}$ax in the implementation paper.)

`max_refinement_steps` ($0 \leq$ integer) 10
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.

`mehrotra_algorithm` (`no, yes`) `no`
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".

`no` Do the usual Ipopt algorithm.

`yes` Do Mehrotra's predictor-corrector algorithm.

`min_hessian_perturbation` ($0 \leq$ real) 10^{-20}
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$\hat{m}$in in implementation paper.)

`min_refinement_steps` ($0 \leq$ integer) 1
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.)

`neg_curv_test_tol` ($0 <$ real) 0
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.

`perturb_always_cd (no, yes)` `no`
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.

`no` perturbation only used when required

`yes` always use perturbation

`perturb_dec_fact` ($0 <$ real < 1) 0.333333
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.)

`perturb_inc_fact` ($1 <$ real) 8
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.)

`perturb_inc_fact_first` ($1 <$ real) 100
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.)

`residual_improvement_factor` ($0 <$ real) 1
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.

`residual_ratio_max` ($0 <$ real) 10^{-10}
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).

`residual_ratio_singular` ($0 <$ real) 10^{-5}
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.

Warm Start

`warm_start_bound_frac` ($0 <$ real ≤ 0.5) 0.001
same as bound_frac for the regular initializer.

`warm_start_bound_push` ($0 <$ real) 0.001
same as bound_push for the regular initializer.

`warm_start_entire_iterate (no, yes)` `no`
Tells algorithm whether to use the GetWarmStartIterate method in the NLP.

`no` call GetStartingPoint in the NLP

`yes` call GetWarmStartIterate in the NLP

`warm_start_init_point` (`no, yes`) `no`
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.)

`no` do not use the warm start initialization

`yes` use the warm start initialization

`warm_start_mult_bound_push` ($0 <$ real) 0.001
same as mult_bound_push for the regular initializer.

`warm_start_mult_init_max` (real) 10^6
Maximum initial value for the equality multipliers.

`warm_start_same_structure` (`no, yes`) `no`
Indicates whether a problem with a structure identical to the previous one is to be solved.
If "yes" is chosen, then the algorithm assumes that an NLP is now to be solved, whose structure is identical to one that already was considered (with the same NLP object).

`no` Assume this is a new problem.

`yes` Assume this is problem has known structure

`warm_start_slack_bound_frac` ($0 <$ real ≤ 0.5) 0.001
same as slack_bound_frac for the regular initializer.

`warm_start_slack_bound_push` ($0 <$ real) 0.001
same as slack_bound_push for the regular initializer.

6 Optimization Services

OS (**O**ptimization **S**ervices) 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 OS manual [20], the papers [14, 15, 16], and the web sites `http://www.optimizationservices.org` and `https://projects.coin-or.org/OS`.

The OS link in GAMS allows you to convert instances of GAMS models into the OS instance language (OSiL) format and let an Optimization Services Server solve your instances remotely.

6.1 Model requirements

OS supports continuous, binary, and integer variables, linear and nonlinear equations. Special ordered sets, semicontinuous or semiinteger variables, and indicator constraints are currently not supported. Initial values are currently not supported by the GAMS/OS link.

6.2 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.

6.3 Detailed Options Descriptions

readosol (*string*)

Specifies the name of an option file in OSoL format that is given to the OS server. This way it is possible to pass options directly to the solvers interfaced by OS.

writeosil (*string*)

Specifies the name of a file in which the GAMS model instance should be writting in OSiL format.

writeosrl (*string*)

Specifies the name of a file in which the result of a solve process (solution, status, ...) should be writting in OSrL format.

service (*string*)

Specifies the URL of an Optimization Services Server. The GAMS model is converted into OSiL format, send to the server, and the result translated back into GAMS format. Note that by default the server chooses a solver that is appropriate to the model type. You can change the solver with the solver option.

solver (*string*)

Specifies the solver that is used to solve an instance on the OS server.

7 OsiCplex, OsiGlpk, OsiGurobi, OsiMosek, OsiSoplex, OsiXpress

The "bare bone" solver links GAMS/OsiCplex, GAMS/OsiGlpk, GAMS/OsiGurobi, GAMS/OsiMosek, GAMS/OsiSoplex, and GAMS/OsiXpress allow users to solve their GAMS models with GLPK, SoPlex, or a standalone license of CPLEX, GUROBI, MOSEK, or XPRESS. 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 OsiGlpk link has been written by Vivian De Smedt, Braden Hunsaker, and Lou Hafer, the OsiGurobi link has been written primarily by Stefan Vigerske, the OsiMosek link has been written primarily by Bo Jensen, and 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`.

7.1 Model requirements

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.

7.2 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` (except for OsiGlpk), `optcr`, `nodlim`, `iterlim`, and `threads` (except for OsiGlpk and 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. For OsiGlpk, an GAMS option file can be provided. See Section 7.3 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` can 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 MPS format before starting the solution process. 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 analog to the `mipstart` option of the full CPLEX and GUROBI links and the `loadmipsol` option of the full XPRESS link.

7.3 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:

```
CPX_PARAM_MIPEMPHASIS            2
CPX_PARAM_HEURFREQ               42
CPX_PARAM_MIPDISPLAY             4
```

OsiGlpk Options

The OsiGlpk option 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 osiglpk.opt.

```
factorization givens
```

It will cause OsiGlpk to use Givens rotation updates for the factorization. (This option setting might help to avoid numerical difficulties in some cases.)

startalg (*string*)

This option determines whether a primal or dual simplex algorithm should be used to solve an LP or the root node of a MIP.

(default = primal)

primal	Let GLPK use a primal simplex algorithm.
dual	Let GLPK use a dual simplex algorithm.

scaling (*string*)

This option determines the method how the constraint matrix is scaled. Note that scaling is only applied when the presolver is turned off, which is on by default.

(default = meanequilibrium)

off	Turn off scaling.
equilibrium	Let GLPK use an equilibrium scaling method.
mean	Let GLPK use a geometric mean scaling method.
meanequilibrium	Let GLPK use first a geometric mean scaling, then an equilibrium scaling.

pricing (*string*)

Sets the pricing method for both primal and dual simplex.

(default = steepestedge)

textbook	Use a textbook pricing rule.
steepestedge	Use a steepest edge pricing rule.

factorization (*string*)

Sets the method for the LP basis factorization.

If you observe poor performce, then you may try setting the factorization method to forresttomlin.

(default = givens)

forresttomlin	Does a LU factorization followed by Forrest-Tomlin updates. This method is fast, but less stable than others.
bartelsgolub	Does a LU factorization followed by a Schur complement and Bartels-Golub updates. This method is slower than Forrest-Tomlin, but more stable.
givens	Does a LU factorization followed by a Schur complement and Givens rotation updates. This method is slower than Forrest-Tomlin, but more stable.

tol_dual (*real*)

Absolute tolerance used to check if the current basis solution is dual feasible.

(default = 1e-7)

tol_primal (*real)*

Relative tolerance used to check if the current basis solution is primal feasible.

(default = 1e-7)

tol_integer (*real)*

Absolute tolerance used to check if the current basis solution is integer feasible.

(default = 1e-5)

backtracking (*string)*

Determines which method to use for the backtracking heuristic.

(default = bestprojection)

depthfirst	Let GLPK use a depth first search.
breadthfirst	Let GLPK use a breadth first search.
bestprojection	Let GLPK use a best projection heuristic.

presolve (*integer)*

Determines whether the LP presolver should be used.

(default = 1)

0	Turns off the LP presolver.
1	Turns on the LP presolver.

cuts (*integer)*

Determines which cuts generator to use: none, all, or user-defined

(default = 0)

-1	Turn off all cut generators
0	Turn on or off each cut generators separately
1	Turn on all cut generators

covercuts (*integer)*

Whether to enable cover cuts.

(default = 1)

0	Turn off cover cuts
1	Turn on cover cuts

cliquecuts (*integer)*

Whether to enable clique cuts.

(default = 1)

0	Turn off clique cuts
1	Turn on clique cuts

gomorycuts (*integer)*

Whether to enable Gomorys mixed-integer linear cuts.

(default = 1)

0	Turn off gomory cuts
1	Turn on gomory cuts

mircuts (*integer*)

Whether to enable mixed-integer rounding cuts.

(default = 0)

0	Turn off mir cuts
1	Turn on mir cuts

reslim (*real*)

Maximum time in seconds.

(default = GAMS reslim)

iterlim (*integer*)

Maximum number of simplex iterations.

(default = GAMS iterlim)

optcr (*real*)

Relative optimality criterion for a MIP. The search is stoped when the relative gap between the incumbent and the bound given by the LP relaxation is smaller than this value.

(default = GAMS optcr)

OsiGurobi Options

In an OsiCplex option file, each line lists one option setting, where the option name and value are separated by space.

Example:

```
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:

```
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:

```
MIPLOG = 3
HEURFREQ = 2
```

COIN-OR References

[1] K. Abhishek, S. Leyffer, and J.T. Linderoth. FilMINT: An outer-approximation-based solver for nonlinear mixed integer programs. *INFORMS Journal On Computing*, 22(4):555–567, 2010.

[2] Patrick R. Amestoy, Iain S. Duff, Jacko Koster, and Jean-Yves L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1):15–24, 2001.

[3] Patrick R. Amestoy, Abdou Guermouche, Jean-Yves L'Excellent, and Stephane Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[4] P. Belotti. Disjunctive cuts for non-convex MINLP. In Jon Lee and Sven Leyffer, editors, *Mixed-integer nonlinear optimization: Algorithmic advances and applications*, IMA volumes in Mathematics and its Applications. Springer, 2011. to appear.

[5] Pietro Belotti. *COUENNE: a user's manual*. https://projects.coin-or.org/Couenne.

[6] Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4–5):597–634, 2009.

[7] Pierre Bonami, Gérard Cornuéjols, Andrea Lodi, and François Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119:331–352, 2009.

[8] Pierre Bonami and João P. M. Gonçalves. Primal heuristics for mixed integer nonlinear programs. Technical Report RC24639, IBM Research, 2008.

[9] Pierre Bonami, Mustafa Kilinç, and Jeffrey Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In Jon Lee and Sven Leyffer, editors, *Mixed-integer nonlinear optimization: Algorithmic advances and applications*, IMA volumes in Mathematics and its Applications. Springer, 2011. to appear.

[10] Pierre Bonami and Jon Lee. *BONMIN Users' Manual*, 1.3 edition, November 2009. https://projects.coin-or.org/Bonmin.

[11] Pierre Bonami, Andreas Wächter, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, and Nicolas W. Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[12] Marco A. Duran and Ignacio E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[13] Roger Fletcher and Sven Leyffer. Solving Mixed Integer Nonlinear Programs by Outer Approximation. *Mathematical Programming*, 66(3(A)):327–349, 1994.

[14] Robert Fourer, Jun Ma, and Kipp Martin. Optimization services: A framework for distributed optimization. *Operations Research*, accepted, 2009. http://www.optimizationservices.org/.

[15] Robert Fourer, Jun Ma, and Kipp Martin. OSiL: An instance language for optimization. *Computational Optimization and Applications*, 45(1):181–203, 2010.

[16] Horand Gassmann, Jun Ma, Kipp Martin, and Wayne Sheng. *Optimization Services 2.1 User's Manual*, March 2010. https://projects.coin-or.org/OS.

[17] Omprakash K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[18] Yoshiaki Kawajir, Carl Laird, and Andreas Wächter. *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt*, 1597 edition, November 2009. https://projects.coin-or.org/Ipopt.

[19] Robin Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, 2003. http://www.coin-or.org.

[20] Jun Ma. *Optimization Services (OS)*. PhD thesis, Industrial Engineering and Management Sciences, Northwestern University, 2005.

[21] Ignacio Quesada and Ignacio E. Grossmann. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & Chemical Engineering*, 16:937–947, 1992.

[22] Andreas Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, January 2002.

[23] Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. http://projects.coin-or.org/Ipopt.

CONOPT

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

Contents

1 Introduction

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. Currently, there are three families of NLP algorithms available, CONOPT, MINOS and SNOPT, and CONOPT is available in three versions, the old CONOPT1 and CONOPT2 and the new CONOPT3.

All algorithms attempt to find a local optimum. The algorithms in CONOPT, MINOS, and SNOPT are all based on fairly different mathematical algorithms, and they behave differently on most models. This means that while CONOPT is superior for some models, MINOS or SNOPT will be superior for others. To offer modelers with a large portfolio of NLP models the best of all worlds, GAMS offers various NLP package deals consisting of two or three NLP solvers for a reduced price if purchased together.

Even CONOPT1, CONOPT2 and CONOPT3 behave differently; the new CONOPT3 is best for most models, but there are a small number of models that are best solved with the older versions and they are therefore still distributed together with CONOPT3 under the same license. However, you should notice that the older versions are no longer being developed, so if you encounter problems with CONOPT1 or CONOPT2, please try to use CONOPT3 instead.

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 NLP solver. If you select CONOPT it implies the default version of CONOPT which from GAMS distribution 21.0 is CONOPT3. If you want to use a different algorithm or algorithm version or 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, `NLP = <solvername>`, on the GAMS command line, or you may rerun 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 MINOS 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 either MINOS or SNOPT 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/CONOPT3 can use second derivatives. If the number of variables is much larger than the number of constraints CONOPT3 (but not CONOPT1 and CONOPT2) will use second derivatives and overall progress can be considerably faster than for MINOS or SNOPT.

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 20000 equations and variables have been solved successfully, and CNS models with over 500000 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, there are no guaranties whatsoever for this class of models and you will often 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.

If possible, you should try to reformulate a DNLP model to an equivalent or approximately equivalent form as described in section 7.

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 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 2), and explain the termination messages (section 3). We will then discuss function evaluation errors (section 4), the use of options (section 5), and give a CONOPT perspective on good model formulation including topics such as initial values and bounds, simplification of expressions, and scaling (section 6). Finally, we will discuss the difference between NLP and DNLP models (section 7). The text is mainly concerned with the new CONOPT3 but most of it will also cover the older versions of CONOPT and we will use the generic name CONOPT when referring to the solver. Some features are only available in the latest CONOPT3 or in CONOPT2 and CONOPT1 in which case we will mention it explicitly. Messages from the older versions of CONOPT may have a format that is slightly different from the one shown here.

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:

```
    C O N O P T 3   Windows NT/95/98  version 3.01F-011-046
    Copyright (C)   ARKI Consulting and Development A/S
                    Bagsvaerdvej 246 A
                    DK-2880 Bagsvaerd, Denmark

 Using default options.

 Reading data

   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

   Iter Phase Ninf   Infeasibility   RGmax    NSB   Step InItr MX OK
    160   0   124  9.5022881759E-01              0.0E+00      F  F
    170   0   134  9.5022881759E-01              0.0E+00      F  F
    180   0   144  9.5022881759E-01              0.0E+00      F  F
    190   0   154  9.5022881759E-01              0.0E+00      F  F
    201   1   160  9.4182618946E-01 4.3E+01   134 2.4E-06      T  T
    206   1   130  8.2388503304E-01 9.5E+01   138 1.0E+00   13 T  T
    211   1    50  1.0242911941E-01 6.9E+00    84 7.2E-01   24 T  T
    216   1    16  2.6057507770E-02 1.3E+00    52 6.1E-01   17 T  T
    221   1     5  7.2858773666E-04 6.1E-03    38 6.0E-01    7 F  F

 ** Feasible solution. Value of objective =    1.00525015566

   Iter Phase Ninf     Objective     RGmax    NSB   Step InItr MX OK
    226   3        1.0092586645E+00 4.4E-04    38 1.0E+00    3 T  T
    231   3        1.0121749760E+00 1.4E+00    24 4.8E-01    9 T  T
    236   3        1.0128148550E+00 4.8E-06    13 5.8E-02   12 F  T
    241   3        1.0128161551E+00 2.5E-06    12 9.1E+03      F  T
    246   4        1.0128171043E+00 1.2E-07    13 1.0E+00    3 F  T
    247   4        1.0128171043E+00 5.7E-08    13

 ** Optimal solution. Reduced gradient less than tolerance.
```

The first few lines identify the version of CONOPT that you use and tell whether you are using an options file or not.

The first few iterations have a special interpretation: iteration 0 represents the initial point exactly as received from GAMS, iteration 1 represent the initial point after 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 "Phase" in column 2. The model is infeasible during Phase 0, 1, and 2 and the Sum of Infeasibilities in column 4 is minimized; the model is feasible during Phase 3 and 4 and the actual objective function, also shown in column 4, 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 Phase 0 iterations. During Phase 1 and 3 the model behaves almost linearly and special linear iterations that take advantage of the linearity are performed, sometimes augmented with some inner "Sequential Linear Programming" (SLP) iterations, indicated by the number of SLP iterations in the InItr column. During Phase 2 and 4 the model behaves more nonlinearly 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 simple models CONOPT will approximate second order information as a byproduct of the line searches. For more complex models CONOPT3 will use some inner "Sequential Quadratic Programming" (SQP) iterations based on exact second derivatives. These inner iterations are identified by the number of 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 gradient of the non-optimal variables. Rgmax should eventually converge towards zero. The last two columns labeled MX and OK gives information about the line search: MX = T means that the line search was terminated by a variable reaching a bound, and MX = F means that the optimal step length was determined by nonlinearities. OK = T means that the line search was well-behaved, and OK = F means that the line search was terminated because it was not possible to find a feasible solution for large step lengths.

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 around 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. 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 (Intermediate Nonoptimal) 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 7. The problem can also be caused by a poorly scaled model. See section 6.5 for hints on model scaling. Finally, it can be caused by stalling as described in section A13.4 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.

```
      <var>: The variable has reached infinity

   ** 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 and it returns with Modelstat = 3 (Unbounded). Check whether the solution appears unbounded or the problem is caused by the scaling of the unbounded variable <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 return a feasible solution with large values for the variables.

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

```
      <var>: Free variable becomes too large

   ** Infeasible solution. A free variable exceeds the allowable
      range.  Current value is 4.20E+07 and current upper bound
      (Rtmaxv) is 3.16E+07
```

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 <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 (Locally Nonoptimal).

```
   ** The iteration limit has been reached.
```

The iteration limit defined in GAMS, either by default (usually 100000 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 (Locally Nonoptimal).

```
   ** 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>sOMLIM = 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 (Locally Nonoptimal). See section 4 for more details on "Function Evaluation Errors".

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

   <var> appearing in
   <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.

   <var> appearing in
   <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 with changes 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 6.5. 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 A4 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 "`<model>.WORKFACTOR = x.x;`" "`<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%
               Derivative evaluations        0.000 =  0.0%

   Work length =   0.35 Mbytes
      Estimate =   0.35 Mbytes
      Max used =   0.35 Mbytes
```

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 = 9 (Error Setup 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 10 (Error Internal Solver Failure) and Modelstat will be either 6 (Intermediate Infeasible) or 7 (Intermediate Nonoptimal) and CONOPT will return primal solution values. The marginals of both equations and variables will be zero or EPS.

The first set of statistics in the message text shows you how much memory is available for CONOPT, and the last set shows how much is available for GAMS and CONOPT combined (GAMS needs space to store the nonlinear functions). 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 combined memory, measured in Mbytes.

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 bounds on your variables. 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 6.3 on "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 that the system crashes. 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 A4 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`" when the solver is CONOPT and "`conopt2.opt`" for the older CONOPT2. 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 is different from the format of options file used by MINOS and ZOOM. It consists in its simplest form of a number of lines like these:

```
rtmaxv = 1.e8
lfnsup = 500
```

Upper case letters are converted to lower case so the second line could also be written as "`LFNSUP = 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, yes`, or `1`, and the logical values representing false are `false, no`, or `0`.

In previous versions of CONOPT you could add "SET" in front of the option assignment. This is no longer supported.

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) .. PM(IT) =E= PWM(IT)*ER*(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)*ER.L*(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 A4 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 A4.3 in Appendix A by adding the line "`lstcrs=t`" to the CONOPT options file (not availabel with CONOPT1). 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) .. X(I) =L= Y;
```

or

```
YSUM   .. Y =E= 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 A4 in Appendix A gives another example of bounds that can be counter productive.

6.3 Simple Expressions

The following model component

```
PARAMETER MU(I);
VARIABLE  X(I), S(I), OBJ;
EQUATION  OBJDEF;
OBJDEF .. OBJ =E= EXP( SUM( I, SQR( X(I) - MU(I) ) / S(I) ) );
```

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

```
PARAMETER MU(I);
VARIABLE  X(I), S(I), OBJ, INTERM;
EQUATION  INTDEF, OBJDEF;
INTDEF .. INTERM =E= SUM( I, SQR( X(I) - MU(I) ) / S(I) );
OBJDEF .. OBJ    =E= EXP( INTERM );
```

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 CONOPT3 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:

```
VARIABLE X(I), Y;
EQUATION YDEF;
YDEF ..  Y =E= 1 / SUM(I, X(I) );
```

should be written as

```
VARIABLE X(I), XSUM, Y;
EQUATION XSUMDEF, YDEF;
XSUMDEF .. XSUM =E= SUM(I, X(I) );
YDEF    .. Y =E= 1 / XSUM;
XSUM.LO = 1.E-2;
```

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

```
CON(I) .. f( SUM( J$(ORD(J) LE ORD(I)), INV(J) ) ) =L= B(I);
```

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

```
CDEF(I) .. CAP(I) =E= INV(I) + CAP(I-1);
```

and the original constraints become

```
CON(I) .. f( 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*(N+1)/2 nonlinear elements to 3*N-1 linear elements and only N nonlinear element. If f is an invertable increasing function you may even rewrite the last constraint as a simple bound:

```
CAP.LO(I) = 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 A4 in Appendix A) will aggregate parts of the model if it is useful for CONOPT without increasing the complexity in GAMS.

6.4 Equalities vs. Inequalities

A resource constraint or a production function is often modeled as an inequality constraint in an optimization model; the optimization algorithm will search over the space of feasible solutions, and if the constraint turns out to constrain the optimal solution the algorithm will make it a binding constraint, and it will be satisfied as an equality. If you know from the economics or physics of the problem that the constraint must be binding in the optimal solution then you have the choice of defining the constraint as an equality from the beginning. The inequality formulation gives a larger feasible space which can make it easier to find a first feasible solution. The feasible space may even be convex. On the other hand, the solution algorithm will have to spend time determining which constraints are binding and which are not. The trade off will therefore depend on the speed of the algorithm component that finds a feasible solution relative to the speed of the algorithm component that determines binding constraints.

In the case of CONOPT, the logic of determining binding constraints is slow compared to other parts of the system, and you should in general make equalities out of all constraints you know must be binding. You can switch to inequalities if CONOPT has trouble finding a first feasible solution.

6.5 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.

6.5.1 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 6.3 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:

```
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:

```
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

```
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

```
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.

6.5.2 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 a follows: The scale factor on a variable, V^s is used to related the variable as seen by the modeler, V^m, to the variable as seen by the algorithm, V^a, as follows:

$$V^m = V^a * V^s$$

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 related the equation as seen by the modeler, G^m, to the equation as seen by the algorithm, G^a, as follows:

$$G^m = G^a * G^s$$

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:

$$dG^a/dV^a = dG^m/dV^m * V^s/G^s$$

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

```
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:

```
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:

```
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 `CARD(I) = CARD(J) = CARD(K) = 100` while `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) = expression;
```

The statement

```
   X.SCALE(I,J,K) = 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 ABS(x1+x2) + 5*ABS(x1-x2) 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 = ABS(f(x)) is replaced by z = fplus + fminus, fplus and fminus are declared as positive variables and they are defined with the identity: f(x) =E= fplus - fminus. 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 fplus = 5, fminus = 0, and z = 5, and with fplus = 1000, fminus = 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 fplus * fminus =e= 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 =E= MAX(f(x),g(y)) is replace by the two inequalities, z =G= f(x) and z =G= 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 =E= MIN(f(x),g(y)) is replaced by the two inequalities, z =L= f(x) and z =L= 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 =E= SMAX(I, f(x,I)) is replaced by the indexed inequality: Ineq(I) .. z =L= 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 ABS(f(x)) is

```
  SQRT( SQR(f(x)) + SQR(delta) )
```

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 SQR(delta)/2 for f(x) = 1. The second derivative is 1/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 ABS exactly when f(x) = 0, then you may use the alternative approximation

```
   SQRT( SQR(f(x)) + SQR(delta) ) - 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 ABS. The Huber function is defined as

```
   H(x) =  x for x  >   delta,
   H(x) = -x for x  <  -delta and
   H(x) = SQR(x)/2/delta + delta/2 for -delta  <  x  <  delta.
```

Although the Huber function has some nice properties, it is for example accurate when ABS(x) > 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 MAX(f(x),g(y)) is

```
   ( f(x) + g(y) + SQRT( SQR(f(x)-g(y)) + SQR(delta) ) )/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

```
   ( f(x) + g(y) + SQRT( SQR(f(x)-g(y)) + SQR(delta) ) - delta )/2
```

Similar smooth GAMS approximations for MIN(f(x),g(y)) are

```
   ( f(x) + g(y) - SQRT( SQR(f(x)-g(y)) + SQR(delta) ) )/2
```

and

```
   ( f(x) + g(y) - SQRT( SQR(f(x)-g(y)) + SQR(delta) ) + delta )/2
```

Appropriate delta values are the same as for the ABS approximation: in the range from 1.e-2 to 1.e-4.

It appears that there are no simple symmetric extensions for MAX and 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 ABS, MAX, or 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 ABS, MAX, or 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 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 =E= const * Q * ABS(Q)**(P-1), where the sign of the Q-term takes care of the sign of dH, and the ABS function guaranties that the real power ** 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 const * (P-1) * ABS(Q)**(P-1) for Q $>$ 0 and -const * (P-1) * 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:

```
Sigmoid(x) = exp( min(x,0) ) / (1+exp(-abs(x)))
```

The standard definition of the sigmoid function is

```
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:

```
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.

```
Sigmoid(x) = (exp(x)/(1+exp(x)))$(x lt 0) + (1/(1+exp(-x)))$(x gt 0)
```

but a $ -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 = SQRT(SQR(x)). The composite function is equivalent to y = 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, SQRT(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 SQRT(SQR(x) + SQR(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 Conic Constraints with GAMS/CONOPT

Certain types of conic constraints can be formulated in GAMS as described in the GAMS/MOSEK user's guide. The GAMS/CONOPT interface translates these constraints into nonlinear constraints and treats them as described in this note.

The quadratic cone is described in GAMS as

```
   Qcone.. x =C= sum(i, y(i) );
```

and it represents the convex nonlinear constraint

```
   x > sqrt( sum(i, sqr( y(i) ) ) ).
```

The rotated quadratic (or hyperbolic) cone is described in GAMS as

```
   Hcone.. x1 + x2 =C= sum(i, y(i) );
```

and it represents the convex nonlinear constraint

```
   sqrt(2*x1*x2) > sqrt( sum(i, sqr( y(i) ) ) ) with x1 > 0 and x2 > 0.
```

The cones are in GAMS/CONOPT implemented using one of two mathematical forms. The mathematical form is selected from the CONOPT option `GCForm` as follows:

`GCForm = 0` (the default value):

```
   QCone.. sqr(x)  =G= sum(i, sqr( y(i) ) );
   Hcone.. 2*x1*x2 =G= sum(i, sqr( y(i) ) );
```

`GCForm = 1`:

```
   QCone.. x+GCPtb2 =G= sqrt( GCPtb1+sum(i, sqr( y(i) ) ) );
   Hcone.. Sqrt( GCPtb1 + 2*x1*x2 ) =G= Sqrt( GCptb1+sum(i, sqr( y(i) ) ) );
```

where `GCPtb1` and `GCPtb2` are perturbation parameters (explained below).

The advantages and disadvantages of the two formulations are as follows: With `GCForm = 0` all functions are quadratic with a sparse Hessian and bounded second derivatives. However, function values grow with `sqr(x)` and first derivatives grow with `x` and CONOPT's automatic scaling methods will sometimes have problems selecting good scaling factors for these equations. With `GCForm = 1` the functions are more complicated with dense Hessian matrices. However, the function values grow linearly with `x` and the first derivatives are unit vectors which usually gives a nicely scaled model.

Although Conic constraints are convex and therefore usually are considered nice they have one bad property, seen from an NLP perspective: The derivatives and/or the dual variables are not well defined at the origin, `y(i) = 0`, because certain constraint qualifications do not hold. With `GCForm = 0` and `x = 0` the constraint is effectively `sum(i, sqr(y(i) ) ) =E= 0` that only has the solution `y(i) = 0`. Since all derivatives are zero the constraint seems to vanish, but if it still is binding the dual variable will go towards infinity, causing all kinds of numerical problems. With `GCForm = 1` the first derivatives do not vanish in the same way. The `y`-part of the derivative vector is a unit vector, but its direction becomes undefined at `y(i) = 0` and the second derivatives goes towards infinity.

The CONOPT option `GCPtb1` is a perturbation used to make the functions smooth around the origin. The default value is 1.e-6 and there is a lower bound of 1.e-12. The `GCPtb1` smoothing increases the value of the right hand side, making the constraint tighter around the origin with a diminishing effect for larger `y`-values. `GCPtb2` is used to control the location of the largest effect of the perturbation. With `GCPtb2 = 0` (the default value) the constraint is tightened everywhere with the largest change of `sqrt(GCPtb1)` around the origin. With `GCPtb2 = sqrt(GCPtb1)` the constraint will go through the origin but will be relaxed with up to `GCPtb2` far from the origin. For many convex model `GCPtb2 = 0` will be a good value. However, models in which it is important the `x = 0` is feasible, e.g. models with binary variables and constraints of the form `x =L= C*bin GCPtb2` must be defined as `sqrt(GCPtb1)`.

The recommendation for selecting the various Conic options is therefore:

- If you expect the solution to be away from the origin then choose the default `GCForm = 0`.
- If the origin is a relevant point choose `GCForm = 1`. If the model is difficult to solve you may try to solve it first with a large value of `GCPtb1`, e.g. 1.e-2, and then re-solve it once or twice each time with a smaller value.
- If you have selected `GCForm = 1`, select `GCPtb2 = sqrt(GCPtb1)` if it is essential that `x = 0` is feasible. Otherwise select the default `GCPtb2 = 0`.

The variables appearing in the Cone constraints are initialized as any other NLP variables, i.e. they are initialized to zero, projected on the bounds if appropriate, unless the modeler has selected other values. Since Cone constraints often behave poorly when y(i) = 0 it is a good idea to assign sensible non-zero values to y(i). The x-values are less critical, but it is also good to assign x-values that are large enough to make the constraints feasible. If you use `GCForm = 1`, remember that the definition of feasibility includes the perturbations.

9 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.

A1 Overview of GAMS/CONOPT

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

```
min or max      f(x)            (1)
subject to      g(x) = b        (2)
lo < x < up                     (3)
```

where `x` is the vector of optimization variables, `lo` and `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. 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, 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.

A2 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_s and possibly on some second order information.
8. Perform a line search along the direction d. For each step, x_s is changed in the direction d_s 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 A3 to A6. The optimizing steps are specialized in several versions according to the 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 A7 and A8). If the model is "very" linear an improved search direction (step 7) can be computed using specialized inner LP-like iterations (section A9), and a steepest edge procedure can be useful for certain models that needs very many iterations (section A10). 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 A11).

The remaining two sections give some short guidelines for selecting non-default options (section A12), and discuss miscellaneous topics (section A13) such as CONOPT's facilities for strictly triangular models (A13.1) and for square systems of equations, in GAMS represented by the model class called CNS or Constrained Nonlinear Systems (A13.2), as well as numerical difficulties due to loss of feasibility (A13.3) and slow or no progress due to stalling (A13.4).

A3 Iteration 0: The Initial Point

The first few "iterations" in the iteration log (see section 2 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 intermediated 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:

```
=E= constraints: ROUND(ABS(EQ.L - EQ.LO),3)
=L= constraints: ROUND(MIN(0,EQ.L - EQ.UP),3)
=G= constraints: ROUND(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 5.e-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.

A4 Iteration 1: Preprocessing

Iteration 1 corresponds to a pre-processing step. Constraints-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, the equations and variables are ordered as shown in Fig. 7.1.

A4.1 Preprocessing: Pre-triangular Variables and Constraints

The pre-triangular equations are those labeled A in Fig.7.1 . 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

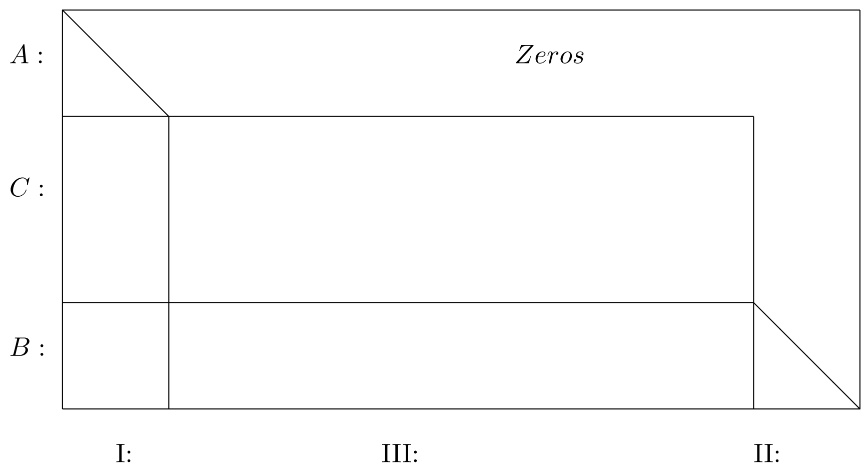


Figure 7.1: The ordered Jacobian after Preprocessing.

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 A13.1.

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

```
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 nonlinearly 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=           9.86122887E-02
   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 =  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 SQR(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 $\texttt{X1}^2$ 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, `rtnwtr` - the triangular Newton tolerance, and if the infeasibility is small it will also tell how the tolerance can be relaxed. Section 5 in the main text on "`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.

A4.2 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 fig.7.1. 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 LSQSMALL / ALL /;
SOLVE LSQSMALL 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, `LSQSMALL`. 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 `LSQSMALL` 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 2 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

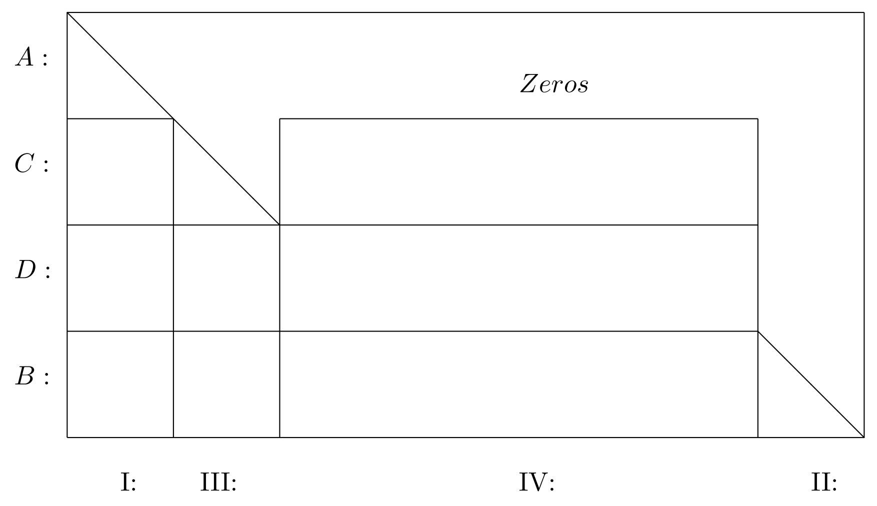


Figure 7.2: The ordered Jacobian after Preprocessing and Crashing.

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.

A4.3 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.

Fig.7.2 shows a reordered version of fig.7.1. 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 Fig. 7.2. There is, as already shown with the small example in section A4.2 above, no guarantie 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.

The Crash procedure is not available in CONOPT1.

A5 Iteration 2: Scaling

Iteration 2 is the last dummy iteration during which the model is scaled, if scaling is turned on. The default in CONOPT3 is to turn scaling on and the default in CONOPT2 is to turn scaling off. There is no scaling in CONOPT1. 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 from CONOPT3 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 "`lsscal = 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 6.5 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 Fig.7.1) 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 abs(Jac(i,j)*X(j)) where 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 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 around 1.e6 so terms much larger than one million will still remain large.

3. The terms 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 variables are divided by this 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 1.e6) are only partially scaled down.

To avoid difficulties with rapidly varying variables and derivatives CONOPT keeps moving averages of the variables and derivatives and uses these averages instead of the variables and derivatives themselves in the scaling procedure described above. It also recomputes the scale factors at regular intervals (see `lfscal`).

The CR-Cells that control scaling, `lsscal`, `lfscal`, `rtmins`, and `rtmaxs`, are all described in Appendix B.

A6 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:

a) 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.

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

In case a) 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 b) 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 b) 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 a) 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 6.1 on "Initial Values".

The start and the iterations of Phase 0 can, in addition to the crash option described in section A6, be controlled with the three CR-cells `lslack`, `lsmxbs`, and `lmmxsf` described in Appendix B.

A7 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 phase 1 (linear mode) and 2 (nonlinear mode) is similar to the distinction between phase 3 and 4 that is described in the next sections.

A8 Linear and Nonlinear Mode: Phase 1 to 4

The optimization itself follows step 2 to 9 of the GRG algorithm shown in A2 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 refered 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.

A9 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 "`lseslp = f`" in the CONOPT options file (usually *conopt.opt*). The default value of `lseslp` (`lseslp` = 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 `lseslp`, 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.

The SLP procedure is not available in CONOPT1.

A10 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, "`lsanrm = t`", in the CONOPT options file (usually *conopt.opt*). The default value of `lsanrm` (`lsanrm` = Logical Switch for A- `NoRM`) is `f` or `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.

The Steepest Edge procedure is not available in CONOPT1.

A11 Nonlinear Mode: The SQP Procedure

When progress is determined by nonlinearities the old CONOPT2 would often take many small steps with small variations in the size of the superbasis and small variations in the reduced gradient. Second order information was necessary to make good progress and to determine if bounds should be active or not. The second order information was estimated over many iterations, but it was often invalidated by basis changes when bounds became active and it had to be estimated again.

In contrast CONOPT3 can use exact second order information about the functions and this information can now be computed by GAMS. The second order information is used in a Sequential Quadratic Programming (SQP) procedure that much like the SLP procedure described above finds a good search direction and a good basis; 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 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. This number 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 ”`lsesqp = f`” in the CONOPT options file (usually `conopt.opt`). The default value of `lsesqp` (lsesqp = Logical Switch Enabling SQP mode) is `t` or `true`, i.e. the SQP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define `lsesqp`, but it can be used for experimentation.

The SQP procedure is only available in CONOPT3.

In connection with 1st and 2nd derivatives the listing file (*.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. CONOPT3 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:

```
Second order sparsety pattern was not generated.
The Hessian of the Lagrangian became too dense because of equation obj.
You may try to increase Rvhess from its default value of 10.
```

CONOPT3 has interrupted the creation of the matrix of second derivatives because it became too dense. A dense matrix of second derivatives will need more memory than CONOPT3 initially has allocated for it, and it may prevent CONOPT3 from performing the optimization with default memory allocations. In addition, it is likely that a dense Hessian will make the SQP iterations so slow that the potential saving in number of iterations is used up computing and manipulating the Hessian.

GAMS/CONOPT3 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.

In this particular case, the offending GAMS equation is "obj". You may consider rewriting this equation. Look for nonlinear functions applied to long expressions such as log(sum(i,x(i)); as discussed in section 6.3. 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 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. Add the option `Rvhess = XX` to the CONOPT options file.

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

```
 CONOPT time Total                          0.734 seconds
   of which: Function evaluations           0.031 =  4.3%
             1st Derivative evaluations     0.020 =  2.7%
             2nd Derivative evaluations     0.113 = 15.4%
             Directional 2nd Derivative     0.016 =  2.1%
```

The function evaluations and 1st derivatives are similar to those reported by CONOPT2. 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 CONOPT3 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 CONOPT not to generate it using option `Dohess = f`.

A12 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 "`lseslp = 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`.

A13 Miscellaneous Topics

A13.1 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 "A4.1 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 4.1) 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, for example *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.

A13.2 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, CONOPT2 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 aCNS 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 tothe non-fixed variables using Newton's method. The solution process will stop with an error message and the current intermediate infeasible solution will be returned if the Jacobian to be inverted is singular, or if one of the non-fixed variables tries to move outside their bounds.

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 guarantie 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 and there cannot be a unique solution to the model. 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 guarantie 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.

A model with these two constraints and the bound

```
   e1 .. x1 + x2 =e= 2;
   e2 .. x1 - x2 =e= 0;
   x1.lo = 1.5;
```

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, (`x1`,`x2`) = (1,1) violates the bound on `x1`. This error case also be combined with model status 5-Locally Infeasible. In the cases where CONOPT2 can guarantie 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 `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:
http://www.gams.com/docs/document.htm

A13.3 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 A6. 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.

A13.4 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 A13.3 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 looses 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 A8). 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`.

A13.5 External Functions

CONOPT1, CONOPT2 and CONOPT3 can be used with external functions written in a programming language such as Fortran or C. CONOPT3 can also use Hessian time vector products from the external functions. Additional information is available at GAMS's web site at http://www.gams.com/docs/extfunc.htm.

Note that CONOPT3 has a Function and Derivative Debugger. Since external functions are dangerous to use CONOPT3 will automatically turned the Function and Derivative Debugger on in the initial point if the model uses external functions. After verifying that you external functions have 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 functions do not depend on other variables than the ones you have specified in the GAMS representation. Structural errors found by these check are usually cased by programming mistakes and must be corrected. The second type of check verifies that the derivatives returned by the external functions are consistent with changes 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). The check is correct if second derivatives are less than `Rtmxj2`. `Rtmxj2` has a default value of 1.e4. You may increase it. However, you are probably going to have solution problems if you have models with second derivatives above 1.e4.

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

10 APPENDIX B - CR-Cells

The CR-Cells that ordinary GAMS users can access are listed below. CR-Cells starting on R assume real values, CR-Cells starting on LS assume logical values (TRUE, T, FALSE, or F), and all other CR-Cells starting on L assume integer values. Several CR-Cells are only used in several versions of CONOPT in which case it will be mentioned below. However, these CR- Cells can still be defined in an options file for the old CONOPT, but they will silently be ignored:

Option	**Description**	**Default**
`lfilog`	Iteration Log frequency. A log line is printed to the screen every `lfilog` iterations (see also `lfilos`). The default value depends on the size of the model: it is 10 for models with less than 500 constraints, 5 for models between 501 and 2000 constraints and 1 for larger models. The log itself can be turned on and off with the Logoption (LO) parameter on the GAMS call.	
`Lfilos`	Iteration Log frequency for SLP and SQP iterations. A log line is printed to the screen every `lfilos` iterations while using the SLP or SQP mode. The default value depends on the size of the model: it is 1 for large models with more than 2000 constraints or 3000 variables, 5 for medium sized models with more than 500 constraints or 1000 variables, and 10 for smaller models.	
`lfderr`	The Function and Derivative Debugger (by default used with external equations) will not write more than `lfderr` error messages independent of the number of errors found.	`10`
`lfmxns`	Limit on new superbasics. When there has been a sufficient reduction in the reduced gradient in one subspace, CONOPT tests if any nonbasic variables should be made superbasic. The ones with largest reduced gradient of proper sign are selected, up to a limit of `lfmxns`. The default value of `lfmxns` is 5. The limit is replaced by the square root of the number of structural variables if `lfmxns` is set to zero.	`5`
`lfnicr`	Limit for slow progress / no increase. The optimization is stopped with a "Slow Progress" message if the change in objective is less than 10 * `rtobjr` * max(1,abs(FOBJ)) for `lfnicr` consecutive iterations where FOBJ is the value of the current objective function. The default value of `lfnicr` is 12.	`12`
`lfnsup`	Maximum Hessian dimension. If the number of superbasics exceeds `lfnsup` CONOPT will no longer store a Reduced Hessian matrix. CONOPT2 will switch to a steepest descend approach, independent of the degree of nonlinearity of the model. The default value of `lfnsup` is 500. If `lfnsup` is increased beyond its default value the default memory allocation may not be sufficient, and you may have to include a "`<model>.WORKSPACE = xx.x;`" statement in your GAMS source file, where "model" represent the GAMS name of the model. You should try to increase the value of `lfnsup` if CONOPT performs many iterations in Phase 4 with the number of superbasics (NSB) larger than `lfnsup` and without much progress. The new value should in this case be larger than the number of superbasics. CONOPT3 will also refrain from using a reduced Hessian. However, it can still use second derivatives in combination with a conjugate gradient algorithm. It is usually not a good idea to increase `lfnsup` much beyond its default value of 500 with CONOPT3. The time used to manipulate a very large reduced Hessian matrix is often large compared to the potential reduction in the number of iterations. (Note: CONOPT2 and CONOPT3 react very differently to changes in `lfnsup`.)	`500`
`lfscal`	Frequency for scaling. The scale factors are recomputed after `lfscal` recomputations of the Jacobian. The default value is 20. Not CONOPT1	`20`

Option	Description	Default
`lfstal`	Maximum number of stalled iterations. If `lfstal` is positive then CONOPT will stop with a "No change in objective" message when the number of stalled iterations as defined in section A13.4 exceeds `lfstal` and `lfstal` is positive. The default value of `lfstal` is 100. Not CONOPT1.	`100`
`lkdebg`	Controls the Function and Derivative Debugger. The value 0 indicates that the debugger should not be useed, the value -1 that it should be used in the initial point, and the value +n that it should be used every n'th time the derivatives are computed. The default value is 0, except for models with external equations where it is -1.	
`lmmxsf`	Method for finding the maximal step while searching for a feasible solution. The step in the Newton direction is usually either the ideal step of 1.0 or the step that will bring the first basic variable exactly to its bound. An alternative procedure uses "bending": All variables are moved a step s and the variables that are outside their bounds after this step are projected back to the bound. The step length is determined as the step where the sum of infeasibilities, computed from a linear approximation model, starts to increase again. The advantage of this method is that it often can make larger steps and therefore better reductions in the sum of infeasibilities, and it is not very sensitive to degeneracies. The alternative method is turned on by setting `lmmxsf` to 1, and it is turned off by setting `lmmxsf` to 0. Until the method has received additional testing it is by default turned off. Not CONOPT1.	`0`
`lsismp`	Logical switch for Ignoring Small Pivots. `Lsismp` is only used when `lstria = t`. If `lsismp = t` (default is `f` or `false`) then a triangular equations is accepted even if the pivot is almost zero (less than `rtpivt` for nonlinear elements and less than `rtpiva` for linear elements), provided the equation is feasible, i.e. with residual less than `rtnwtr`. Not CONOPT1.	`false`
`lslack`	Logical switch for slack basis. If `lslack = t` then the first basis after preprocessing will have slacks in all infeasible constraints and Phase 0 will usually be bypassed. This is sometimes useful together with `lstcrs = t` if the number of infeasible constraints after the crash procedure is small. This is especially true if the SLP procedure described in section A9 quickly can remove these remaining infeasibilities. It is necessary to experiment with the model to determine if this option is useful.	
`lsmxbs`	Logical Switch for Maximal Basis. `lsmxbs` determines whether CONOPT should try to improve the condition number of the initial basis (`t` or `true`) before starting the Phase 0 iterations or just use the initial basis immediately (`f` or `false`). The default value is `t`, i.e. CONOPT tries to improve the basis. There is a computational cost associated with the procedure, but it will usually be saved because the better conditioning will give rise to fewer Phase 0 iterations and often also to fewer large residuals at the end of Phase 0. The option is ignored if `lslack` is true. Not CONOPT1.	`true`
`lspost`	Logical switch for the Post-triangular preprocessor. If `lspost = f` (default is `t` or `true`) then the post-triangular preprocessor discussed in section A4.2 is turned off.	`true`
`lspret`	Logical switch for the Pre-triangular preprocessor. If `lspret = f` (default is `t` or `true`) then the pre-triangular preprocessor discussed in section A4.1 is turned off.	`true`
`lsscal`	Logical switch for scaling. A logical switch that turns scaling on (with the value `t` or `true`) or off (with the value `f` or `false`). The default value is `f`, i.e. no scaling for CONOPT2 and `t`, i.e. scaling for CONOPT3. It is not available with CONOPT1.	`false`
`lssqrs`	Logical switch for Square Systems. If `lssqrs = t` (default is `f` or `false`), then the model must be a square system as discussed in section A13.2. Users are recommended to use the CNS model class in GAMS. Not CONOPT1.	`false`

Option	Description	Default
`lstria`	Logical switch for triangular models. If `lstria = t` (default is `f` or `false`) then the model must be triangular as discussed in section A13.1. Not CONOPT1.	`false`
`rtmaxj`	Maximum Jacobian element. The optimization is stopped if a Jacobian element exceeds this value. `rtmaxj` is initialized to a value that depends on the machine precision. It is on most machines around 2.5. The actual value is shown by CONOPT in connection with "Too large Jacobian element" messages. If you need a larger value then your model is poorly scaled and CONOPT may find it difficult to solve.	
`rtmaxv`	Internal value of infinity. The model is considered unbounded if a variable exceeds `rtmaxv` in absolute value. `rtmaxv` is initialized to a value that depends on the machine precision. It is on most machines around 6.e7. The actual value is shown by CONOPT in connection with "Unbounded" messages. If you need a larger value then your model is poorly scaled and CONOPT may find it difficult to solve.	
`rtmaxs`	Scale factors larger than `rtmaxs` are rounded down to `rtmaxs`. The default value is `1024` in CONOPT2 and `1024*1024` in CONOPT3.	`1024*1024`
`rtmxj2`	Upper bound on second derivatives used by the Function and Derivative Debugger to determine if a derivative computed by the modeler is consistent with a numerically computed derivative.	`1.e4`
`rtminj`	All Jacobian elements with a value less than `rtminj` are rounded up to the value `rtminj` before scaling is started to avoid problems with zero and very small Jacobian elements. The default value is 1.e-5. Only CONOPT2.	`1.e-5`
`rtmins`	Scale factors smaller than `rtmins` are rounded up to `rtmins`. The default value is in CONOPT2 and `1/1024` in CONOPT2 and `1`. (All scale factors are powers of 2 to avoid round-off errors from the scaling procedure).	`1`
`rtnwma`	Maximum feasibility tolerance. A constraint will only be considered feasible if the residual is less than `rtnwma` times MaxJac, independent on the dual variable. MaxJac is an overall scaling measure for the constraints computed as max(1,maximal Jacobian element/100). The default value of `rtnwma` is 1.e-3.	`1.e-3`
`rtnwmi`	Minimum feasibility tolerance. A constraint will always be considered feasible if the residual is less than `rtnwmi` times MaxJac (see above), independent of the dual variable. The default value depends on the machine precision. It is on most machines around 4.e-10. You should only increase this number if you have inaccurate function values and you get an infeasible solution with a very small sum of infeasibility, or if you have very large terms in some of your constraints (in which case scaling may be more appropriate). Square systems (see `lssqrs` and section A13.2) are always solved to the tolerance `rtnwmi`.	
`rtnwtr`	Triangular feasibility tolerance. If you solve a model, fix some of the variables at their optimal value and solve again and the model then is reported infeasible in the pre-triangular part, then you should increase `rtnwtr`. The infeasibilities in some unimportant constraints in the "Optimal" solution have been larger than `rtnwtr`. The default value depends on the machine precision. It is on most machines around 6.e-7.	
`rtobjr`	Relative objective tolerance. CONOPT assumes that the reduced objective function can be computed to an accuracy of `rtobjr` * max(1,abs(FOBJ)) where FOBJ is the value of the current objective function. The default value of `rtobjr` is machine specific. It is on most machines around 3.e-13. The value is used in tests for "Slow Progress", see `lfnicr`.	
`rtoned`	Relative accuracy of one-dimensional search. The one-dimensional search is stopped if the expected further decrease in objective estimated from a quadratic approximation is less than `rtoned` times the decrease obtained so far. The default value is 0.2. A smaller value will result in more accurate but more expensive line searches and this may result in an overall decrease in the number of iterations. Values above 0.7 or below 0.01 should not be used.	`0.2`

Option	Description	Default
rtpiva	Absolute pivot tolerance. A pivot element is only considered acceptable if its absolute value is larger than `rtpiva`. The default value is 1.e-10. You may have to decrease this value towards 1.e-11 or 1.e-12 on poorly scaled models.	1.e-10
rtpivr	Relative pivot tolerance. A pivot element is only considered acceptable relative to other elements in the column if its absolute value is at least `rtpivr` * the largest absolute value in the column. The default value is 0.05. You may have to increase this value towards one on poorly scaled models. Increasing `rtpivr` will result in denser `L` and `U` factors of the basis.	0.05
rtpivt	Triangular pivot tolerance. A nonlinear triangular pivot element is considered acceptable if its absolute value is larger than `rtpivt`. The default value is 1.e-7. Linear triangular pivot must be larger than `rtpiva`.	1.e-7
rtredg	Optimality tolerance. The reduced gradient is considered zero and the solution optimal if the largest superbasic component is less than `rtredg`. The default value depends on the machine, but is usually around 9.e-8. If you have problems with slow progress or stalling you may increase `rtredg`. This is especially relevant for very large models.	9.e-8
rvspac	A space allocation factor that sometime can speed up the solution of square systems. CONOPT will tell you if it is worth while to set this parameter to a non-default value for your class of model.	
rvstlm	Step length multiplier. The step length in the one-dimensional line search is not allowed to increased by a factor of more than `rvstlm` between steps for models with nonlinear constraints and a factor of 100 * `rvstlm` for models with linear constraints. The default value is 4.	4
dohess	A logical variable that controls the creation of the Hessian (matrix of second derivatives). The default value depends on the model. If the number of equalities is very close to the number of non-fixed variables then the solution is assumed to be in a corner point or in a very low dimensional space where second derivatives are not needed, and `dohess` is initialized to false. Otherwise `dohess` is initialized to true. If `dohess` is false you will not get statistics about the Hessian in the listing file. It takes some time to generate second order information and it uses some space. If CONOPT3 generates this information for your model but it does not use it, i.e. if you see that no time is spend on 2nd derivative evaluations, then you may experiment with `dohess` turned off. If the number of Hessian elements is very large you may also try turning `dohess` off. Note that CONOPT3 still can use directional second derivatives and therefore use its SQP algorithm in the cases where the Hessian is not available. (CONOPT3 only).	
rvhess	A real number that controls the space available for creation of the Hessian. The maximum number of nonzero elements in the Hessian and in some intermediate terms used to compute it is limited by `Rvhess` times the number of Jacobian elements (first derivatives). The default value of `Rvhess` is 10, which means that the Hessian should not be denser than 10 second derivatives per first derivative. (CONOPT3 only).	10
Gcform	Defines the functional form used to implement Cone constraints as a nonlinear inequality constraint using a 0-1 value. `0`: The Cone constraints are implemented as `sqr(x) =G= sum(i, sqr( y(i) ) )` for the quadratic cone and `2*x1*x2 =G= sum(i, sqr( y(i) ) )` for the rotated or hyperbolic cone. `1`: The cone constraints are implemented as `x+GCPtb2 =G= sqrt(GCPtb1+sum(i,sqr(y(i))))` for the quadratic cone and `(GCPtb1+2*x1*x2) =G= Sqrt(GCptb1+sum(i,sqr(y(i))))` for the rotated or hyperbolic cone.	0

Option	Description	Default
`Gcptb1`	A perturbation used to smooth Cone constraints around the origin and ensure that derivatives are defined. The lower bound is 1.e-12. Is only used when `Gcform=1`.	`1.e-6`
`Gcptb2`	A perturbation that can be used to force the smoothed Cone constraints through the origin. Is only used when `Gcform=1`. The perturbation is bounded above by `sqrt(Gcptb1)`.	

11 APPENDIX C: References

J. Abadie and J. Carpentier, Generalization of the Wolfe Reduced Gradient Method to the case of Nonlinear Constraints, in Optimization, R. Fletcher (ed.), Academic Press, New York, 37-47 (1969).

A. Drud, A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems, Mathematical Programming 31, 153-191 (1985).

A. S. Drud, CONOPT - A Large-Scale GRG Code, ORSA Journal on Computing 6, 207- 216 (1992).

A. S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Tutorial for CONOPT Subroutine Library, 16p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark (1995).

A. S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Reference Manual for CONOPT Subroutine Library, 69p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark (1996).

J. K. Reid, A Sparsity Exploiting Variant of Bartels-Golub Decomposition for Linear Programming Bases, Mathematical Programming 24, 55-69 (1982).

D. M. Ryan and M. R. Osborne, On the Solution of Highly Degenerate Linear Programmes, Mathematical Programming 41, 385-392 (1988).

U. H. Suhl and L. M. Suhl, Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases, ORSA Journal on Computing 2, 325-335 (1990).

CONVERT

Contents

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 idenifiable 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 the following formats:

- AlphaECP
- AMPL
- AmplNLC
- BARON
- CplexLP
- CplexMPS
- Dict
- FixedMPS
- GAMS (Scalar format)
- Jacobian
- LAGO
- LGO
- LindoMPI

- LINGO
- MINOPT
- NLP2MCP
- ViennaDag

For more information see the options section.

2 How to use CONVERT

CONVERT is run like any other GAMS solver. From the command line this is:

```
>> 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:

```
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, ..., em`.

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 `trnsport.gms` into scalar format, One would run `gams trnsport.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 =G= 325;
e5..    x2 + x5 =G= 300;
e6..    x3 + x6 =G= 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.

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 then look for and read an option file with the name *convert.opt* (see "Using Solver Specific Options" 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 table.

Option	Description	Default
`all`	Generates all supported file formats.	
`AlphaECP`	Generates AlphaECP input file.	`alpha.ecp`
`Ampl`	Generates AMPL input file.	`ampl.mod`
`AmplNLC`	Generates Ampl NLC compatible file.	`amplnlc.c`
`Baron`	Generates BARON input file.	`gams.bar`
`ConeReform`	Reformulation of cone `=C=` constraints to NLP format. `0`: keep conic `=C=` format `1`: convert conic constraints to NLP format	`0`
`CplexLP`	Generates CPLEX LP format input file.	`cplex.lp`
`CplexMPS`	Generates CPLEX MPS format input file.	`cplex.mps`
`Dict`	Convert to GAMS dictionary.	`dict.txt`
`FileList`	Generates file list of file formats generated.	`file.txt`
`FixedMPS`	Generates fixed format MPS file.	`fixed.mps`
`Gams`	Generates GAMS scalar model. This is the default conversion format used.	`gams.gms`
`Jacobian`	Writes GDX version of current point.	`jacobian.gdx`
`GmsInsert`	Inserts the line `$if NOT '%gams.u1%' == '' $include'%gams.u1%'` before the solve statement.	
`help`	Generates option summary.	
`include <filename>`	Start reading from a new file.	
`Lago`	Generates a partial Lago file.	`lago.gms`
`Lgo`	Generates an LGO FORTRAN file.	`lgomain.for`
`LindoMPI`	Generates Lindo MPI file.	`lindo.mpi`
`Lingo`	Generates Lingo input file.	`lingo.lng`
`match`	Force a complete match for all MCP variable / equation pairs.	
`memo`	Generates a memo file containing model statistics and files created .	`memo.txt`
`Minopt`	Generates Minopt input file.	`minopt.dat`
`NLP2MCP`	Generates GAMS scalar MCP model.	`gamsmcp.gms`
`ObjVar`	Name of objective variable. By default the objective variable is just named via index, for example `x1`.	
`Reform`	Force reformulations.	`100`
`Terminate`	Force GAMS to terminate after conversion.	
`ViennaDag`	Generates Vienna Dag input file.	`vienna.dag`

CPLEX 12

Contents

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 varaibles 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 reformuated 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 `solnpoolagap` 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 *diversity filter*.
- 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.

3.5.1 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 `SolnPoolPopRepeat`) 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 `SolnPoolPopRepeat` determined to continue the search for alternative solutions, the file specified by option `SolnPoolPopDel` 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.

3.5.2 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 `EpInt`
- the feasibility tolerance `EpRHS`

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.

3.5.3 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*.

3.5.4 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.

3.5.5 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.

3.5.6 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:

```
...
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;
);
```

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.

Simlarly, 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

$$(|BP - BF|)/(1.0e - 10 + |BF|) < \text{OptCR}$$

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:

$$(|BP - BF|)/(|BP|) < \text{OptCR}$$

***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

advind	advanced basis use
aggfill	aggregator fill parameter
aggind	aggregator on/off
clocktype	clock type for computation time
coeredind	coefficient reduction on/off
depind	dependency checker on/off
feasopt	computes a minimum-cost relaxation to make an infeasible model feasible
feasoptmode	Mode of FeasOpt
.feaspref	feasibility preference
interactive	allow interactive option setting after a Control-C
lpmethod	algorithm to be used for LP problems
memoryemphasis	Reduces use of memory
names	load GAMS names into Cplex
numericalemphasis	emphasizes precision in numerically unstable or difficult problems

objrng	do objective ranging
parallelmode	parallel optimization mode
predual	give dual problem to the optimizer
preind	turn presolver on/off
prelinear	linear reduction indicator
prepass	number of presolve applications to perform
printoptions	list values of all options to GAMS listing file
qpmethod	algorithm to be used for QP problems
reduce	primal and dual reduction type
relaxpreind	presolve for initial relaxation on/off
rerun	rerun problem if presolve infeasible or unbounded
rhsrng	do right-hand-side ranging
rngrestart	write GAMS readable ranging information file
scaind	matrix scaling on/off
solutiontarget	type of solution when solving a nonconvex continuous quadratic model
threads	global default thread count
tilim	overrides the GAMS ResLim option
tuning	invokes parameter tuning tool
tuningdisplay	level of information reported by the tuning tool
tuningmeasure	measure for evaluating progress for a suite of models
tuningrepeat	number of times tuning is to be repeated on perturbed versions
tuningtilim	tuning time limit per model or suite
workdir	directory for working files
workmem	memory available for working storage

5.2 Simplex Algorithmic Options

craind	crash strategy (used to obtain starting basis)
dpriind	dual simplex pricing
epper	perturbation constant
iis	run the IIS finder if the problem is infeasible
netfind	attempt network extraction
netppriind	network simplex pricing
perind	force initial perturbation
perlim	number of stalled iterations before perturbation
ppriind	primal simplex pricing
pricelim	pricing candidate list
reinv	refactorization frequency

5.3 Simplex Limit Options

itlim	iteration limit
netitlim	iteration limit for network simplex
objllim	objective function lower limit
objulim	objective function upper limit
singlim	limit on singularity repairs

5.4 Simplex Tolerance Options

epmrk	Markowitz pivot tolerance
epopt	optimality tolerance
eprhs	feasibility tolerance
netepopt	optimality tolerance for the network simplex method
neteprhs	feasibility tolerance for the network simplex method

5.5 Barrier Specific Options

baralg	algorithm selection
barcolnz	dense column handling
barcrossalg	barrier crossover method
barepcomp	convergence tolerance
bargrowth	unbounded face detection
baritlim	iteration limit
barmaxcor	maximum correction limit
barobjrng	maximum objective function
barorder	row ordering algorithm selection
barqcpepcomp	convergence tolerance for the barrier optimizer for QCPs
barstartalg	barrier starting point algorithm

5.6 Sifting Specific Options

siftalg	sifting subproblem algorithm
siftitlim	limit on sifting iterations

5.7 MIP Algorithmic Options

bbinterval	best bound interval
bndstrenind	bound strengthening
brdir	set branching direction
bttol	backtracking limit
cliques	clique cut generation
covers	cover cut generation
cutlo	lower cutoff for tree search
cuts	default cut generation
cutsfactor	cut limit
cutup	upper cutoff for tree search
disjcuts	disjunctive cuts generation
divetype	MIP dive strategy
eachcutlim	Sets a limit for each type of cut
flowcovers	flow cover cut generation
flowpaths	flow path cut generation
fpheur	feasibility pump heuristic
fraccuts	Gomory fractional cut generation
gubcovers	GUB cover cut generation

heurfreq	heuristic frequency
implbd	implied bound cut generation
lbheur	local branching heuristic
mcfcuts	multi-commodity flow cut generation
mipemphasis	MIP solution tactics
mipkappastats	MIP kappa computation
mipordind	priority list on/off
mipordtype	priority order generation
mipsearch	search strategy for mixed integer programs
mipstart	use mip starting values
miqcpstrat	MIQCP relaxation choice
mircuts	mixed integer rounding cut generation
nodefileind	node storage file indicator
nodesel	node selection strategy
preslvnd	node presolve selector
probe	perform probing before solving a MIP
qpmakepsdind	adjust MIQP formulation to make the quadratic matrix positive-semi-definite
relaxfixedinfeas	access small infeasibilties in the solve of the fixed problem
repeatpresolve	reapply presolve at root after preprocessing
rinsheur	relaxation induced neighborhood search frequency
solvefinal	switch to solve the problem with fixed discrete variables
startalg	MIP starting algorithm
strongcandlim	size of the candidates list for strong branching
strongitlim	limit on iterations per branch for strong branching
subalg	algorithm for subproblems
submipnodelim	limit on number of nodes in an RINS subMIP
symmetry	symmetry breaking cuts
varsel	variable selection strategy at each node
zerohalfcuts	zero-half cuts

5.8 MIP Limit Options

aggcutlim	aggrigation limit for cut generation
auxrootthreads	number of threads for auxiliary tasks at the root node
cutpass	maximum number of cutting plane passes
fraccand	candidate limit for generating Gomory fractional cuts
fracpass	maximum number of passes for generating Gomory fractional cuts
intsollim	maximum number of integer solutions
nodelim	maximum number of nodes to solve
polishafterepagap	Absolute MIP gap before starting to polish a feasible solution
polishafterepgap	Relative MIP gap before starting to polish a solution
polishafternode	Nodes to process before starting to polish a feasible solution
polishafterintsol	MIP integer solutions to find before starting to polish a feasible solution
polishaftertime	Time before starting to polish a feasible solution
probetime	time spent probing
repairtries	try to repair infeasible MIP start
trelim	maximum space in memory for tree

5.9 MIP Solution Pool Options

divfltup	upper bound on diversity

divfltlo	lower bound on diversity
.divflt	solution pool range filter coefficients
populatelim	limit of solutions generated for the solution pool by populate method
readflt	reads Cplex solution pool filter file
solnpool	solution pool file name
solnpoolagap	absolute tolerance for the solutions in the solution pool
solnpoolcapacity	limits of solutions kept in the solution pool
solnpoolgap	relative tolerance for the solutions in the solution pool
solnpoolintensity	solution pool intensity for ability to produce multiple solutions
solnpoolpop	methods to populate the solution pool
solnpoolpopdel	file with solution numbers to delete from the solution pool
solnpoolpoprepeat	method to decide if populating the solution should be repeated
solnpoolprefix	file name prefix for GDX solution files
solnpoolreplace	strategy for replacing a solution in the solution pool
userincbcall	The GAMS command line to call the incumbent checking program

5.10 MIP Tolerance Options

epagap	absolute stopping tolerance
epgap	relative stopping tolerance
epint	integrality tolerance
objdif	overrides GAMS Cheat parameter
relobjdif	relative cheat parameter

5.11 Output Options

bardisplay	progress display level
clonelog	enable clone logs
mipdisplay	progress display level
mipinterval	progress display interval
mpslongnum	MPS file format precision of numeric output
netdisplay	network display level
quality	write solution quality statistics
siftdisplay	sifting display level
simdisplay	simplex display level
writebas	produce a Cplex basis file
writeflt	produce a Cplex solution pool filter file
writelp	produce a Cplex LP file
writemps	produce a Cplex MPS file
writemst	produce a Cplex mst file
writeord	produce a Cplex ord file
writeparam	produce a Cplex parameter file with all active options
writepre	produce a Cplex LP/MPS/SAV file of the presolved problem
writesav	produce a Cplex binary problem file

5.12 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-continous and semi-integer variables. These variable types are directly supported by GAMS/Cplex. For example:

```
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-continous 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 candiates for relaxation and weigthed 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:

```
   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:

```
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 relaxtion 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:

  Rows in Factor           = 243
  Integer space required   = 578
  Total non-zeros in factor = 8491
  Total FP ops to factor   = 410889

 Itn      Primal Obj        Dual Obj  Prim Inf Upper Inf  Dual Inf
   0  -1.2826603e+06   7.4700787e+08  2.25e+10  6.13e+06  4.00e+05
   1  -2.6426195e+05   6.3552653e+08  4.58e+09  1.25e+06  1.35e+05
   2  -9.9117854e+04   4.1669756e+08  1.66e+09  4.52e+05  3.93e+04
   3  -2.6624468e+04   2.1507018e+08  3.80e+08  1.04e+05  1.20e+04
   4  -1.2104334e+04   7.8532364e+07  9.69e+07  2.65e+04  2.52e+03
   5  -9.5217661e+03   4.2663811e+07  2.81e+07  7.67e+03  9.92e+02
   6  -8.6929410e+03   1.4134077e+07  4.94e+06  1.35e+03  2.16e+02
   7  -8.3726267e+03   3.1619431e+06  3.13e-07  6.84e-12  3.72e+01
   8  -8.2962559e+03   3.3985844e+03  1.43e-08  5.60e-12  3.98e-02
   9  -3.8181279e+03   2.6166059e+03  1.58e-08  9.37e-12  2.50e-02
  10  -5.1366439e+03   2.8102021e+03  3.90e-06  7.34e-12  1.78e-02
  11  -1.9771576e+03   1.5960442e+03  3.43e-06  7.02e-12  3.81e-03
  12  -4.3346261e+02   8.3443795e+02  4.99e-07  1.22e-11  7.93e-04
  13   1.2882968e+02   5.2138155e+02  2.22e-07  1.45e-11  8.72e-04
  14   5.0418542e+02   5.3676806e+02  1.45e-07  1.26e-11  7.93e-04
  15   2.4951043e+02   6.5911879e+02  1.73e-07  1.43e-11  5.33e-04
  16   2.4666057e+02   7.6179064e+02  7.83e-06  2.17e-11  3.15e-04
  17   4.6820025e+02   8.1319322e+02  4.75e-06  1.78e-11  2.57e-04
  18   5.6081604e+02   7.9608915e+02  3.09e-06  1.98e-11  2.89e-04
  19   6.4517294e+02   7.7729659e+02  1.61e-06  1.27e-11  3.29e-04
  20   7.9603053e+02   7.8584631e+02  5.91e-07  1.91e-11  3.00e-04
  21   8.5871436e+02   8.0198336e+02  1.32e-07  1.46e-11  2.57e-04
  22   8.8146686e+02   8.1244367e+02  1.46e-07  1.84e-11  2.29e-04
  23   8.8327998e+02   8.3544569e+02  1.44e-07  1.96e-11  1.71e-04
  24   8.8595062e+02   8.4926550e+02  1.30e-07  2.85e-11  1.35e-04
  25   8.9780584e+02   8.6318712e+02  1.60e-07  1.08e-11  9.89e-05
  26   8.9940069e+02   8.9108502e+02  1.78e-07  1.07e-11  2.62e-05
  27   8.9979049e+02   8.9138752e+02  5.14e-07  1.88e-11  2.54e-05
  28   8.9979401e+02   8.9139850e+02  5.13e-07  2.18e-11  2.54e-05
  29   9.0067378e+02   8.9385969e+02  2.45e-07  1.46e-11  1.90e-05
  30   9.0112149e+02   8.9746581e+02  2.12e-07  1.71e-11  9.61e-06
  31   9.0113610e+02   8.9837069e+02  2.11e-07  1.31e-11  7.40e-06
  32   9.0113661e+02   8.9982723e+02  1.90e-07  2.12e-11  3.53e-06
  33   9.0115644e+02   9.0088083e+02  2.92e-07  1.27e-11  7.35e-07
  34   9.0116131e+02   9.0116262e+02  3.07e-07  1.81e-11  3.13e-09
  35   9.0116154e+02   9.0116154e+02  4.85e-07  1.69e-11  9.72e-13
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.

        Nodes                                        Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0        0.0000    24                      0.0000       40
*     0+    0        6.0000     0        6.0000        0.0000       40  100.00%
*    50+   50        4.0000     0        4.0000        0.0000      691  100.00%
    100    99        2.0000    15        4.0000        0.4000     1448   90.00%
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*)

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*)

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 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)***

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)***

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 = -1)

-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)***

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)***

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)***

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)***

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)***

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)***

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)***

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)***

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)***

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)***

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)***

Range: [1e-012,1e+075]

(default = 1e-007)

barstartalg ***(integer)***

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)***

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)***

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)***

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)*

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)

cliques (*integer)*

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)*

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.

(default = 0)

0	Automatic
1	CPU time
2	Wall clock time

clonelog (*integer)*

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 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

covers *(integer)*

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)*

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)*

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: $obj \geq c$.

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)*

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)*

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, 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*)

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*)

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: $obj \leq c$.

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*)

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

disjcuts (*integer*)

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*)

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

divfltup (*real*)

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)

divfltlo (*real*)

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)

.divflt (*real*)

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)

dpriind (*integer*)

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*)

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 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 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. 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*)

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*)

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-009,0.1]

(default = 1e-006)

epper (*real*)

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-006)

eprhs (*real*)

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-009,0.1]

(default = 1e-006)

feasopt (*integer*)

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*)

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*)

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*)

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)***

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)***

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)***

Limits the number of candidate variables for generating Gomory fractional cuts.

(default = 200)

fraccuts ***(integer)***

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)***

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)***

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*)

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*)

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*)

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*)

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.

(default = 0)

intsollim (*integer*)

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.

(default = large)

itlim (*integer*)

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*)

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

lpmethod ***(integer)***

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. The solution is returned by first method to finish.

(default = 0)

0	Automatic
1	Primal Simplex
2	Dual Simplex
3	Network Simplex
4	Barrier
5	Sifting
6	Concurrent

mcfcuts ***(integer)***

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)***

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)***

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*)

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

mipkappastats (*integer*)

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

mipinterval (*integer*)

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)

mipordind ***(integer)***

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)***

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)***

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)***

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)***

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***)

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***)

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***)

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***)

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***)

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***)

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*)

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 the network simplex method.

(default = large)

netppriind (*integer*)

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*)

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*)

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*)

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)*

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)*

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.

(default = 0)

objllim *(real)*

Setting a lower objective function limit will cause Cplex to halt the optimization process once the minimum objective function value limit has been exceeded.

(default = -1e+075)

objrng *(string)*

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.

(default = no objective ranging is done)

objulim *(real)*

Setting an upper objective function limit will cause Cplex to halt the optimization process once the maximum objective function value limit has been exceeded.

(default = 1e+075)

parallelmode *(integer)*

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 explicitely specified.

If *parallelmode* is explicitely 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 is list of possible value:

(default = 1)

-1	Enable opportunistic parallel search mode
0	Automatic
1	Enable deterministic parallel search mode

perind (*integer*)

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*)

Perturbation limit. The number of stalled iterations before perturbation is invoked. The default value of 0 means the number is determined automatically.

(default = 0)

polishafterepagap (*real*)

Solution polishing can yield better solutions in situations where good solutions are otherwise hard to find. More time-intensive than other heuristics, solution polishing is actually a variety of branch-and-cut that works after an initial solution is available. In fact, it requires a solution to be available for polishing, either a solution produced by branch-and-cut, or a MIP start supplied by a user. Because of the high cost entailed by solution polishing, it is not called throughout branch-and-cut like other heuristics. Instead, solution polishing works in a second phase after a first phase of conventional branch-and-cut. As an additional step after branch-and-cut, solution polishing can improve the best known solution. As a kind of branch-and-cut algorithm itself, solution polishing focuses solely on finding better solutions. Consequently, it may not prove optimality, even if the optimal solution has indeed been found. Like the RINS heuristic, solution polishing explores neighborhoods of previously found solutions by solving subMIPs.

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)

polishafterepgap (*real*)

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)

polishafternode (*integer*)

Sets the number of nodes processed in branch-and-cut before CPLEX starts solution polishing, if a feasible solution is available.

(default = 2100000000)

polishafterintsol (*integer*)

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)

polishaftertime (*real*)

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 = 0)

populatelim (*integer*)

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 solnpoolcapacity 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*)

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*)

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*)

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, if 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*)

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 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*)

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*)

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)

printoptions (*integer*)

Write the values of all options to the GAMS listing file. Valid values are no or yes.

(default = 0)

probe (*integer*)

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

probetime (*real*)

Limits the amount of time in seconds spent probing.

(default = 1e+075)

qpmakepsdind (*integer*)

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*)

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

quality ***(integer)***

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)

readflt ***(string)***

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)***

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. 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)***

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)***

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)***

The relative version of the objdif option. Ignored if objdif is non-zero.

(default = 0)

repairtries ***(integer)***

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)***

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)***

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)***

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)***

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 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)***

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)***

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)***

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)***

Sets the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.

(default = large)

simdisplay ***(integer)***

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)***

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)***

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)***

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 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 *solnpoolcapacity* are generated. Instead, stopping criteria are regular node and time limits and populatelim, solnpoolgap and solnpoolagap.

(default = 2100000000)

solnpoolgap ***(real)***

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*)

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.

solnpoolpop (*integer*)

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*)

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 delete 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)***

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)***

(default = soln)

solnpoolreplace ***(integer)***

(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)***

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.

solvefinal ***(integer)***

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)***

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)***

Limit on the length of the candidate list for strong branching (varsel = 3).

(default = 10)

strongitlim ***(integer)***

Limit on the number of iterations per branch in strong branching (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)***

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)***

Controls the number of nodes explored in an RINS subMIP. See option rinsheur.

(default = 500)

symmetry ***(integer)***

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)***

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: $\max(1, \text{number of cores} - |\texttt{threads}|)$

(default = GAMS Threads)

tilim ***(real)***

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)***

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 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`. Tuning is incompatible with the BCH facility and other advanced features of GAMS/Cplex.

tuningdisplay ***(integer)***

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)***

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)***

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)***

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*GAMS ResLim)*

userincbcall ***(string)***

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)*

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)*

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 = current or project directory)

workmem *(real)*

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)*

Write a basis file.

writeflt *(string)*

Write the diversity filter to a Cplex FLT file.

writelp *(string)*

Write a file in Cplex LP format.

writemps *(string)*

Write an MPS problem file.

writemst *(string)*

Write a Cplex mst (containing the mip start) file.

writeord *(string)*

Write a Cplex ord (containing priority and branch direction information) file.

writeparam *(string)*

Write a Cplex parameter (containing all modified Cplex options) file.

writepre *(string)*

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)***

Write a binary problem file.

zerohalfcuts ***(integer)***

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

DECIS

Gerd Infanger; Vienna University of Technology; Stanford University

Contents

[0] Copyright © 1989 – 1999 by Gerd Infanger. All rights reserved. The GAMS/DECIS User's Guide is copyrighted and all rights are reserved. Information in this document is subject to change without notice and does not represent a commitment on the part of Gerd Infanger. The DECIS software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement. The DECIS software can be licensed through Infanger Investment Technology, LLC or through Gams Development Corporation.

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) [5], 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) [4]. 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 1.2 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 2 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 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.

1.2 What DECIS Can Do

DECIS solves two-stage stochastic linear programs with recourse:

$$\begin{array}{rrcrcrl}
\min\ z & = & cx & + & E\, f^\omega y^\omega & & \\
s/t & & Ax & & & = & b \\
 & & -B^\omega x & + & D^\omega y^\omega & = & d^\omega \\
 & & x, & & y^\omega & \geq & 0, \quad \omega \in \Omega.
\end{array}$$

where x denotes the first-stage, y^ω the second-stage decision variables, c represents the first-stage and f^ω the second-stage objective coefficients, A, b represent the coefficients and right hand sides of the first-stage constraints, and B^ω, D^ω, d^ω represent the parameters of the second-stage constraints, where the transition matrix B^ω couples the two stages. In the literature D^ω 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 ω with probability $p(\omega)$, where Ω 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^ω is made based on knowledge of the actual outcome ω. 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*:

$$\begin{array}{rrcrcrcccrcl}
\min\ z & = & cx & + & p^1fy^1 & + & p^2fy^2 & + & \cdots & + & p^Wfy^W & \\
s/t & & Ax & & & & & & & & & = & b \\
& & -B^1x & + & Dy^1 & & & & & & & = & d^1 \\
& & -B^2x & & & + & Dy^2 & & & & & = & d^2 \\
& & \vdots & & & & & & \ddots & & & & \vdots \\
& & -B^Wx & & & & & & & + & Dy^W & = & d^W \\
& & x, & & y^1, & & y^2, & & \ldots, & & y^W & \geq & 0,
\end{array}$$

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 folowing representation is also often used:

$$\begin{array}{rcrcl}
\min\ cx & + & E\ z^\omega(x) & & \\
Ax & & & = & b \\
x & & & \geq & 0
\end{array}$$

where

$$\begin{array}{rcl}
z^\omega(x) & = & \min\ f^\omega y^\omega \\
D^\omega y^\omega & = & d^\omega + B^\omega x \\
y^\omega & \geq & 0,\ \omega \in \Omega = \{1, 2, \ldots, W\}.
\end{array}$$

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 $v^\omega = (v_1, \ldots, v_h)^\omega$ 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:

$$f^\omega = f(v^\omega), \quad B^\omega = B(v^\omega), \quad D^\omega = D(v^\omega), \quad d^\omega = d(v^\omega).$$

Each component V_i has outcomes $v_i^{\omega_i}$, $\omega_i \in \Omega_i$, where ω_i labels a possible outcome of component i, and Ω_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

$$\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_h$$

represents the crossing of sets Ω_i. Assuming each set Ω_i contains W_i possible outcomes, $|\Omega_i| = W_i$, the set Ω 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

$$p^\omega = p_1^{\omega_1} p_2^{\omega_2} \cdots p_h^{\omega_h}.$$

Let η denote the vector of all second-stage random parameters, e.g., $\eta = \text{vec}(f, B, D, d)$. The outcomes of η may be represented by the following general linear dependency model:

$$\eta^\omega = \text{vec}(f^\omega, B^\omega, d^\omega, d^\omega) = Hv^\omega, \quad \omega \in \Omega$$

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 Maual. 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,
- the specification of the decision stages,
- the specification of the random parameters, and
- 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 apl1p /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;
omax.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 omax(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.

2.4.1 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 stoch 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 v1, a two-dimensional array of outcomes "out" and corresponding probability "pro" for each of the possible realizations of the set omega1, "o11", "o12", "o13", and "o14". For example, the random parameter v1 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 v1 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 g1 and g2 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 v3, v4, and v5 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
;
```

2.4.2 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 stochasic 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:

```
set s scenario /pessimistic, average, optimistic/;
parameter outcome(s) / pessimistic 1.5
                       average     2.0
                       optimistic  2.3 /;
parameter prob(s)    / pessimistic 0.2
                       average     0.6
                       optimistic  0.2 /;
```

then the correct way of generating the entries in the stochastic file would be:

```
loop(s,
   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 *hm*1 gives the coefficients of the independent random parameter $v1$ in each of the three demand levels and the vector $hm2$ gives the coefficients of the independent random parameter $v2$ 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 *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 *BL* labels a possible realization of an independent random parameter. The name besides the *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., $v1$ and $v2$. 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 *BL v1 period2* indicates that an outcome of (independent random parameter) *v1* is being defined. The name *period2* indicates that it is a second-stage random parameter, and *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 $h1$ for intermediately storing the results of the calculation, looping over the different demand levels *dl* we calculate *h1 = hm1(dl) * v1("out", omega1)* and define the dependent random parameters as the right-hand sides of equation *demand*(*dl*).

When defining an independent random parameter outcome, if the block name is the same as the previous one (e.g., when *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 *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 $omega1$ defines all realizations of the independent random parameter $v1$ and the loop over all elements of $omega2$ defines all realizations of the independent random parameter $v2$.

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;
apl1p.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 *apl1p.optfile = 1* forces GAMS to process the file DECIS.OPT, in which you may define any DECIS parameters.

2.5.1 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.

2.5.2 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.

istrat — Defines the solution strategy used. The default value is istrat = 3.

istrat = 1 Solves the expected value problem. All stochastic parameters are replaced by their expected values and the corresponding deterministic problem is solved using decomposition.

istrat = 2 Solves the stochastic problem using Monte Carlo importance sampling. You have to additionally specify what approximation function you wish to use, and the sample size used for the estimation, see below.

istrat = 3 Refers to istrat = 1 plus istrat = 2. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using importance sampling.

istrat = 4 Solves the stochastic universe problem by enumerating all possible combinations of realizations of the second-stage random parameters. It gives you the exact solution of the stochastic program. This strategy may be impossible, because there may be way too many possible realizations of the random parameters.

istrat = 5 Refers to istrat = 1 plus istrat = 4. First solves the expected value problem using decomposition, then continues and solves the stochastic universe problem by enumerating all possible combinations of realizations of second-stage random parameters.

istrat = 6 Solves the stochastic problem using crude Monte Carlo sampling. No variance reduction technique is applied. This strategy is especially useful if you want to test a solution obtained by using the evaluation mode of DECIS. You have to specify the sample size used for the estimation. There is a maximum sample size DECIS can handle. However, this maximum sample size does not apply when using crude Monte Carlo. Therefore, in this mode you can specify very large sample sizes, which is useful when evaluating a particular solution.

istrat = 7 Refers to istrat = 1 plus istrat = 6. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using crude Monte Carlo sampling.

istrat = 8 Solves the stochastic problem using Monte Carlo pre-sampling. A Monte Carlo sample out of all possible universe scenarios, sampled from the original probability distribution, is taken, and the corresponding "sample problem" is solved using decomposition.

istrat = 9 Refers to istrat = 1 plus istrat = 8. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using Monte Carlo pre-sampling.

istrat = 10 Solves the stochastic problem using control variates. You also have to specify what approximation function and what sample size should be used for the estimation.

istrat = 11 Refers to istrat = 1 plus istrat = 10. First solves the expected value problem using decomposition, then continues and solves the stochastic problem using control variates.

nsamples — Sample size used for the estimation. It should be set greater or equal to 30 in order to fulfill the assumption of large sample size used for the derivation of the probabilistic bounds. The default value is nsamples = 100.

nzrows — Number of rows reserved for cuts in the master problem. It specifies the maximum number of different cuts DECIS maintains during the course of the decomposition algorithm. DECIS adds one cut during each iteration. If the iteration count exceeds nzrows, then each new cut replaces a previously generated cut, where the cut is replaced that has the maximum slack in the solution of the (pseudo) master. If nzrows is specified as too small then DECIS may not be able to compute a solution and stops with an error message.

If nzrows is specified as too large the solution time will increase. As an approximate rule set nzrows greater than or equal to the number of first-stage variables of the problem. The default value is nzrows = 100.

iwrite — Specifies whether the optimizer invoked for solving subproblems writes output or not. The default value is iwrite = 0.

iwrite = 0 No optimizer output is written.

iwrite = 1 Optimizer output is written to the file "MODEL.MO" in the case MINOS is used for solving subproblems or to the file MODEL.CPX in the case CPLEX is used for solving subproblems. The output level of the output can be specified using the optimizer options. It is intended as a debugging device. If you set iwrite = 1, for every master problem and for every subproblem solved the solution output is written. For large problems and large sample sizes the files"MODEL.MO" or "MODEL.CPX" may become very large, and the performance of DECIS may slow down.

ibug — Specifies the detail of debug output written by DECIS. The output is written to the file "MODEL.SCR", but can also be redirected to the screen by a separate parameter. The higher you set the number of ibug the more output DECIS will write. The parameter is intended to help debugging a problem and should be set to ibug = 0 for normal operation. For large problems and large sample sizes the file "MODEL.SCR" may become very large, and the performance of DECIS may slow down. The default value is ibug = 0.

ibug = 0 This is the setting for which DECIS does not write any debug output.

ibug = 1 In addition to the standard output, DECIS writes the solution of the master problem on each iteration of the Benders decomposition algorithm. Thereby it only writes out variable values which are nonzero. A threshold tolerance parameter for writing solution values can be specified, see below.

ibug = 2 In addition to the output of ibug = 1, DECIS writes the scenario index and the optimal objective value for each subproblem solved. In the case of solving the universe problem, DECIS also writes the probability of the corresponding scenario.

ibug = 3 In addition to the output of ibug = 2, DECIS writes information regarding importance sampling. In the case of using the additive approximation function, it reports the expected value for each i-th component of $\bar{\Gamma}_i$, the individual sample sizes N_i, and results from the estimation process. In the case of using the multiplicative approximation function it writes the expected value of the approximation function $\bar{\Gamma}$ and results from the estimation process.

ibug = 4 In addition to the output of ibug = 3, DECIS writes the optimal dual variables of the cuts on each iteration of the master problem.

ibug = 5 In addition to the output of ibug = 4, DECIS writes the coefficients and the right-hand side of the cuts on each iteration of the decomposition algorithm. In addition it checks if the cut computed is a support to the recourse function (or estimated recourse function) at the solution $\hat{x}^k$ at which it was generated. If it turns out that the cut is not a support, DECIS writes out the value of the (estimated) cut and the value of the (estimated) second stage cost at $\hat{x}^k$.

ibug = 6 In addition to the output of ibug = 5, DECIS writes a dump of the master problem and the subproblem in MPS format after having decomposed the problem specified in the core file. The dump of the master problem is written to the file "MODEL.P01" and the dump of the subproblem is written to the file "MODEL.P02". DECIS also writes a dump of the subproblem after the first iteration to the file "MODEL.S02".

iscratch — Specifies the internal unit number to which the standard and debug output is written. The default value is iscratch = 17, where the standard and debug output is written to the file "MODEL.SCR". Setting iscratch = 6 redirects the output to the screen. Other internal unit numbers could be used, e.g., the internal unit number of the printer, but this is not recommended.

ireg — Specifies whether or not DECIS uses regularized decomposition for solving the problem. This option is considered if MINOS is used as a master and subproblem solver, and is not considered if using CPLEX, since regularized decomposition uses a nonlinear term in the objective. The default value is ireg = 0.

rho — Specifies the value of the ρ parameter of the regularization term in the objective function. You will have to experiment to find out what value of rho works best for the problem you want to solve. There is no rule of thumb as to what value should be chosen. In many cases it has turned out that regularized decomposition reduces the iteration count if standard decomposition needs a large number of iterations. The default value is rho = 1000.

tolben — Specifies the tolerance for stopping the decomposition algorithm. The parameter is especially important for deterministic solution strategies, i.e., 1, 4, 5, 8, and 9. Choosing a very small value of tolben may result in a significantly increased number of iterations when solving the problem. The default value is 10^{-7}.

tolw — Specifies the nonzero tolerance when writing debug solution output. DECIS writes only variables whose values are nonzero, i.e., whose absolute optimal value is greater than or equal to tolw. The default value is 10^{-9}.

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"
```

2.5.3 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) [5]. The following parameters should be specified with some consideration:

AIJ TOLERANCE — Specifies the nonzero tolerance for constraint matrix elements 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
```

2.5.4 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”, “FEASIBILIITYTOL”, 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.

FEASIBILIITYTOL — 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.

2.6.1 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 beeing 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:

```
 T H E   D E C I S   S Y S T E M
 Copyright (c) 1989  -- 1999 by Dr. Gerd Infanger
 All rights reserved.

iter            lower          best upper       current upper

   0         -0.9935E+06
   1         -0.4626E+06        0.2590E+05          0.2590E+05
   2          0.2111E+05        0.2590E+05          0.5487E+06
   3          0.2170E+05        0.2590E+05          0.2697E+05
   4          0.2368E+05        0.2384E+05          0.2384E+05
   5          0.2370E+05        0.2384E+05          0.2401E+05
   6          0.2370E+05        0.2370E+05          0.2370E+05

iter            lower          best upper       current upper

   6          0.2370E+05
   7          0.2403E+05        0.2470E+05          0.2470E+05
   8          0.2433E+05        0.2470E+05          0.2694E+05
   9          0.2441E+05        0.2470E+05          0.2602E+05
  10          0.2453E+05        0.2470E+05          0.2499E+05
  11          0.2455E+05        0.2470E+05          0.2483E+05
  12          0.2461E+05        0.2467E+05          0.2467E+05
  13          0.2461E+05        0.2467E+05          0.2469E+05
  14          0.2461E+05        0.2465E+05          0.2465E+05
  15          0.2463E+05        0.2465E+05          0.2467E+05
  16          0.2463E+05        0.2465E+05          0.2465E+05
  17          0.2464E+05        0.2465E+05          0.2465E+05
  18          0.2464E+05        0.2464E+05          0.2464E+05
  19          0.2464E+05        0.2464E+05          0.2464E+05
  20          0.2464E+05        0.2464E+05          0.2464E+05
  21          0.2464E+05        0.2464E+05          0.2464E+05
  22          0.2464E+05        0.2464E+05          0.2464E+05

Normal Exit
```

2.6.2 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.

2.6.3 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.

2.6.4 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 variable) from CPLEX when solving master and sub problems.

A GAMS/DECIS Illustrative Examples

A.1 Example APL1P

```
*   APL1P 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, 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;
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;
```

A.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
* DECISM 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;
```

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) containes 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 incorect. 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.

DECIS References

[1] Brooke, A., Kendrik, D. and Meeraus, A. (1988): *GAMS, A Users Guide*, The Scientific Press, South San Francisco, California.

[2] CPLEX Optimization, Inc. (1989–1997): *Using the CPLEX Callable Library*, 930 Tahoe Blvd. Bldg. 802, Suite 279, Incline Village, NV 89451, USA.

[3] Infanger, G. (1994): *Planning Under Uncertainty – Solving Large-Scale Stochastic Linear Programs*, The Scientific Press Series, Boyd and Fraser.

[4] Infanger, G. (1997): *DECIS User's Guide*, Dr. Gerd Infanger, 1590 Escondido Way, Belmont, CA 94002.

[5] Murtagh, B.A. and Saunders, M.A. (1983): MINOS User's Guide, SOL 83-20, Department of Operations Research, Stanford University, Stanford CA 94305.

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,
  4. 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.

Limited Warranty:

Gerd Infanger warrants that the media on which the Software is distributed will he 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.

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.

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.

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

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 these MINLP optimization problems has not yet reached the stage of maturity and reliability as linear, integer or non-linear programming modeling, these problems have a rich area of applications. 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.

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 and to 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 advised to have access to multiple solvers. Free student/demo systems are available from GAMS Development Corporation. These systems are restricted in the size of models that can be solved.

3 How to Run a Model with GAMS/DICOPT

DICOPT is capable of solving only MINLP models. If you did not specify DICOPT as the default solver, then you can use the following statement in your GAMS model:

```
option minlp = dicopt;
```

It 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 the GAMS statements like:

```
option nlp = conopt;  { or any other nlp solver }
option mip = cplex;   { or any other mip solver }
```

These options can also be specified on the command line, like:

```
> gams mymodel minlp=dicopt nlp=conopt mip=cplex
```

In the IDE (Integrated Development Enviroment) the command line option can be specified in the edit line in the right upper corner of the main window.

Possible NLP solvers include `minos5`, `minos`, `conopt`, `conopt3`, and `snopt`. Possible MIP solvers are `cplex`, `osl`, `osl2`, `osl3`, `xpress`, and `xa`.

With an option file it is even possible to use alternate solvers in different cycles. Section 8 explains this is in detail.

4 Overview of DICOPT

DICOPT solves models of the form:

MINLP	min or max	$f(x,y)$
	subject to	$g(x,y) \sim b$
		$\ell_x \le x \le u_x$
		$y \in \lceil \ell_y \rceil, ..., \lfloor u_y \rfloor$

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 ℓ 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:

$$\begin{aligned} \text{minimize or maximize } & f(x) + c^T y \\ \text{subject to } & G(x) + Hy \sim b \\ & \ell \leq x \leq u \\ & y \in \{0,1\} \end{aligned} \tag{11.1}$$

i.e. the discrete variables are binary variables and they appear linearly in the model.

If we reorder the equations into equality and inequality equations, and convert the problem into a minimization problem, we can write:

$$\begin{aligned} \text{minimize } & c^T y + f(x) \\ \text{subject to } & Ay + h(x) = 0 \\ & By + g(x) \leq 0 \\ & \ell \leq x \leq u \\ & y \in \{0,1\} \end{aligned} \tag{11.2}$$

Let $y^{(0)}$ be any fixed binary vector and let $x^{(0)}$ be the solution of the corresponding NLP subproblem:

$$\begin{aligned} \text{minimize } & c^T y^{(0)} + f(x) \\ \text{subject to } & Ay^{(0)} + h(x) = 0 \\ & By^{(0)} + g(x) \leq 0 \\ & \ell \leq x \leq u \end{aligned} \tag{11.3}$$

Further let

$$\begin{aligned} T^{(0)} &= \operatorname{diag}(t_{i,i}) \\ t_{i,i} &= \operatorname{sign}(\lambda_i) \end{aligned} \tag{11.4}$$

where λ_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:

$$\begin{aligned} \text{minimize } & c^T y^{(0)} + f(x) \\ \text{subject to } & T^{(0)}(Ay^{(0)} + h(x)) \leq 0 \\ & By^{(0)} + g(x) \leq 0 \\ & \ell \leq x \leq u \end{aligned} \tag{11.5}$$

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 the case of 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. This bound can be used as a stopping criterion through a DICOPT option `stop 1` (see section 8). Another stopping criterion that tends to work very well in practice for non-convex problems (and even on 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 penatly 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 $y^{(1)}, y^{(2)}, ..., y^{(K)}$.

For a detailed description of the theory and references to earlier work, see [5, 3, 1].

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, it 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:

```
  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 succesful, 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 can not 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 you have more than one NLP solver available, you may want to try a sequence of them:

```
  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 in 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 really want to solve a MIP sub-problem with an optimality tolerance, you can use the DICOPT option file to set `OPTCR` or `OPTCA` in there. For more information see section 8.

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 with respect to knowledge and experience. A good overview of integer programming modeling is given in [6].

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:

$$\begin{aligned} C_i &= f_i y_i + v_i x_i \\ x_i &\le M_i y_i \\ y_i &\in \{0,1\} \\ 0 \le x_i &\le cap_i \end{aligned} \tag{11.6}$$

where f_i is the fixed cost and v_i the variables cost of operating facility i. In this case M_i should be chosen large enough that x_i is not restricted if $y_i = 1$. On the other hand, we want it as small as possible. This leads to the choice to have M_i equal to the (tight) upperbound of variable x_i (i.e. the capacity cap_i of facility i).

6.4 Non-smooth Functions

NLP modelers are alerted by GAMS against the use of non-smooth functions such as `min()`, `max()`, `smin()`, `smax()` and `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:

```
option dnlp=conopt;
model m /all/;
solve m minimizing z using dnlp;
```

This construct is to 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:

$$f(x) \approx \frac{2x}{1 + e^{-x/h}} - x \tag{11.7}$$

For more information see [2].

For MINLP models, there is not such a protection against non-smooth functions. However, the use of such functions is just as problematic here. However, with MINLP models we have the possibility to use discrete variables, in order to model if-then-else situations. For the case of the absolute value for instance we can replace x by $x^+ - x^-$ and $|x|$ by $x^+ + x^-$ by using:

$$\begin{aligned} x &= x^+ - x^- \\ |x| &= x^+ + x^- \\ x^+ &\le \delta M \\ x^- &\le (1-\delta)M \\ x^+, x^- &\ge 0 \\ \delta &\in \{0,1\} \end{aligned} \tag{11.8}$$

where δ is a binary variable.

7 GAMS Options

GAMS options are specified in the GAMS model source, either using the `option` statement or using 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:

```
   model m /all/;
   option iterlim=100;
   solve m using minlp minimizing z;
```

Here follows a list of option statements that affect the behavior of DICOPT:

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 to occur. In many cases this can be accomplished by adding appropriate bounds. Sometimes you will need to add extra variables and equations to accomplish this. For instance with an expression like $\log(x - y)$, you may want to introduce a variable $z > \varepsilon$ 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 sub-problems. The default is 1000.

option minlp = dicopt;
: 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. The default is 1000 seconds.

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: they 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:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

Here follows a list of model suffices that affect the behavoir of DICOPT:

***m*.dictfile = 1;**
: This option tells GAMS to write a dictionary file containing information about GAMS identifiers (equation and variables names). This information is needed when the DICOPT option `nlptracelevel` is used. Otherwise this 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.,

```
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):

```
--- 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:

```
--- 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 will turn 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 the performance of the MIP solver. 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*.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`. In order to tell DICOPT to read this file, you will need to set the `optfile` model suffix, as in:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

The option file is searched for in the current directory, or in case the IDE (Integrated Development Environment) is used, in the project directory.

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:

```
* 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:

```
$onecho > dicopt.opt
stop 0
nlpsolver minos conopt
$offecho
```

This will make the model self-contained. Notice however that this overwrites an existing file `dicopt.opt`.

Here follows a list of available DICOPT options:

continue n

This option can be used to let DICOPT continue in case of NLP solver failures. The preferred approach is to fix the model, such that NLP subproblems solve without problems. However, in some cases we can ignore (partial) failures of an NLP solver in solving the NLP subproblems as DICOPT may recover later on. During model debugging, you may therefore add the option `continue 0`, in order for DICOPT to function in a more finicky way.

continue 0

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 negligable (indicating convergence) but the reduced gradients are not within the required tolerance. Such a solution may or may not be close the (local) optimum. Using `continue 0` will cause DICOPT not to accept such a solution.

continue 1

NLP subproblem failures resulting in a non-optimal but feasible solutions are accepted. Sometimes an NLP solver can not make further progress towards meeting all optimality conditions, although the current solution is feasible. Such a solution can be accepted by this option.

continue 2

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 $i_1\ i_2\ \ldots\ i_n$

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 `domlim` is used. The last number i_n sets a domain error limit for all cycles $n, n+1, \ldots$.

Example: `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 `domlim` option.

epsmip x

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. Thus, if the options `OPTCR` or `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 OPTCR or OPTCA setting. In that case
you may want to increase the EPSMIP setting using a
DICOPT option file.
```

The value of

$$\frac{\text{PreviousObj} - \text{CurrentObj}}{1 + |\text{PreviousObj}|} \tag{11.9}$$

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 message above, so you will have information about how much you should increase `epsmip` to pass the test. Normally, you should not have to change this value. The default is $x = 1.0e-6$.

epsx x
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 `relaxed`. Default: $x = 1.0e-3$.

infeasder n
This option is to determine whether linearizations of infeasible NLP subproblems are added or not to the MIP master problem.

infeasder 0
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) so as to ensure that the NLP subproblems are mathematically feasible.

infeasder 1
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). If used for a nonconvex MINLP possibility of cutting-off the global optimum is increased.

The default is $n = 0$.

maxcycles n
The maximum number of cycles or major iterations performed by DICOPT. The default is $n = 20$.

mipiterlim $i_1\ i_2\ \ldots\ i_n$
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 can not 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.

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: 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`, the second one to read the option file `mip2.opt` and 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`, `osl2.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 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 can not 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 s_1 s_2 ... s_n

This option specifies with MIP solver to use for the MIP master problems.

Example: `mipsolver cplex osl2`

This instructs DICOPT to use Cplex for the first MIP and OSL2 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 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 $n, n+1, \ldots$. 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 `iterlim`.

Example: `nlpiterlim 1000 -1`

The first (relaxed) NLP subproblem can not 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. The default is not to restrict the amount of iterations an NLP solver can spend on an NLP subproblem, other than the global iteration limit.

nlpoptfile $s_1\ s_2\ \dots\ s_n$

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: 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`, the second one to read the option file `nlp2.opt` and 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 $x_1\ x_2\ \dots\ x_n$

Sets a resource (time) limit on individual NLP subproblems. The last number x_n is valid for all subsequent cycles $n, n+1, \dots$. 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 can not 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. The default for this option is not to restrict individually the time an NLP solver can spend on an NLP subproblem (other than the global resource limit).

nlpsolver $s_1\ s_2\ \dots\ s_n$

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 one uses 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 s

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 n

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 `nlptracefile`.

nlptracelevel 0

No trace files are written. This is the default.

nlptracelevel 1

A GAMS file for each NLP subproblem is written which fixes the discrete variables.

nlptracelevel 2

As `nlptracelevel 1`, but in addition level values of the continuous variables are written.

nlptracelevel 3
As `nlptracelevel 2`, but in addition marginal values for the equations and variables are written.

By including a trace file to 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 10.4).

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. To allow to solve 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 `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

$$|\text{BestFound} - \text{BestPossible}| \le x \tag{11.10}$$

It is possible to specify a different `optca` value for each cycle. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$.

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 x_1 x_2 ... x_n
The relative 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. To allow to solve 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

$$\frac{|\text{BestFound} - \text{BestPossible}|}{|\text{BestPossible}|} \le x \tag{11.11}$$

Note that the relative gap can not be evaluated if the best possible integer solution is zero. In those 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 n
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 can not be solved, but where NLP sub-problems with the integer variables fixed are much easier. In such a case, if a reasonable integer configuration is known in advance, 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) = INITVAL(I);`.

relaxed 0
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.

relaxed 1
The first NLP problem is the relaxed NLP problem: all integer variables are relaxed between their bounds. This is the default.

relaxed 2
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 `EPSX`. `EPSX` has a default value of 1.e-3. This can be changed in through the option file.

stop *n*
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.

stop 0
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 find often the optimal solution using one of the more optimistic stopping rules.

stop 1
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`.

stop 2
Stop as soon as the NLP subproblems stop to improve. This"worsening" criterion is a heuristic. For non-convex problems in which valid bounds can not be obtained the heuristic works often 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.

stop 3
Stop as soon as a crossover occurs or when the NLP subproblems start to worsen. (This is a combination of 1 and 2).

Note: In general a higher number stops earlier, although in some cases stopping rule 2 may terminate the search earlier than rule 1. Section VI shows some experiments with these stopping criteria.

weight *x*
The value of the penalty coefficients. Default $x = 1000.0$.

9 DICOPT Output

DICOPT generates lots of output on the screen. Not only does DICOPT itself writes messages to the screen, but also the NLP and MIP solvers that handle the sub-problems. The most important part is the last part of the screen output.

In this section we will discuss the output that DICOPT writes to the screen and the listing file using the model `procsel.gms` (this model is part of the GAMS model library). A DICOPT log is written there and the reason why DICOPT terminated.

```
--- DICOPT: Checking convergence
--- DICOPT: Search stopped on worsening of NLP subproblems
--- DICOPT: Log File:
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.05        8      0      conopt
  MIP    1         2.48869      0.28        7      0      cplex
  NLP    2         1.72097<     0.00        3      0      conopt
  MIP    2         2.17864      0.22       10      0      cplex
  NLP    3         1.92310<     0.00        3      0      conopt
  MIP    3         1.42129      0.22       12      0      cplex
  NLP    4         1.41100      0.00        8      0      conopt
--- 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
--- PROCSEL.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.

It should be noted 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:

```
               S O L V E     S U M M A R Y

     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.

 ------------------------------------------------------------------
 Dicopt2x-C    Jul  4, 2001 WIN.DI.DI 20.1 026.020.039.WAT
 ------------------------------------------------------------------

     Aldo Vecchietti and Ignacio E. Grossmann
     Engineering Design Research Center
     Carnegie Mellon University
     Pittsburgh, Pennsylvania 15213

     Erwin Kalvelagen
     GAMS Development Corp.
     1217 Potomac Street, N.W.
     Washington DC 20007
 ------------------------------------------------------------------
 DICOPT Log File
 ------------------------------------------------------------------
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.05        8       0      conopt
  MIP    1         2.48869      0.28        7       0      cplex
  NLP    2         1.72097<     0.00        3       0      conopt
  MIP    2         2.17864      0.22       10       0      cplex
  NLP    3         1.92310<     0.00        3       0      conopt
  MIP    3         1.42129      0.22       12       0      cplex
  NLP    4         1.41100      0.00        8       0      conopt
 ------------------------------------------------------------------
   Total solver times :  NLP =     0.05   MIP =     0.72
   Perc. of total     :  NLP =     6.59   MIP =    93.41
 ------------------------------------------------------------------
```

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. Also for each iteration the configuration of the binary variables will 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:
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.06        8       0      conopt
  MIP    1         2.48869      0.16        7       0      cplex
  NLP    2         1.72097<     0.00         3      0      conopt
  MIP    2         2.17864      0.10        10      0      cplex
  NLP    3         1.92310<     0.00         3      0      conopt
  MIP    3         1.42129      0.11        12      0      cplex
  NLP    4         1.41100      0.00         8      0      conopt
  MIP    4         0.00000      0.22        23      0      cplex
  NLP    5         0.00000      0.00         3      0      conopt
  MIP    5        -0.27778      0.16        22      0      cplex
  NLP    6        -0.27778      0.00         3      0      conopt
  MIP    6        -1.00000      0.16        21      0      cplex
  NLP    7        -1.00000      0.00         3      0      conopt
  MIP    7        -1.50000      0.22        16      0      cplex
  NLP    8        -1.50000      0.00         3      0      conopt
  MIP    8        -2.50000      0.11        16      0      cplex
  NLP    9        -2.50000      0.00         3      0      conopt
  MIP    9        *Infeas*      0.11         0      0      cplex
--- 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 finds often 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 anymore. 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 indeed 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 case 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.

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 `stop 1`. This option is however the best stopping criterion for convex problems.

10.2 Solving the NLP Problems

In case the relaxed NLP and/or the other NLP sub-problems are very difficult, using a combination of NLP solvers has been found to be effective. For example, MINOS has much 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 `nlpsolver` 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 can not 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 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:

$$\begin{aligned} \min\ & f(x,y) \\ & g(x,y) \sim b \\ & \ell \le x \le u \\ & y \in \{0,1\} \end{aligned} \tag{11.12}$$

where $\sim$ is a vector of relational operators $\{\le, =, \ge\}$. 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:

$$\begin{aligned} \min\ & f(x,y) + M\sum_i (s_i^+ + s_i^-) \\ & y = y^B + s^+ - s^- \\ & g(x,y) \sim b \\ & \ell \le x \le u \\ & 0 \le y \le 1 \\ & 0 \le s^+, s^- \le 1 \\ & y^B \in \{0,1\} \end{aligned} \tag{11.13}$$

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

When there are many discrete variables, the MIP master problems may become expensive to solve. One of the first thing 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 to spend an unacceptable long time in 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 as 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. Especially CONOPT 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=CONOPT3;`" to solve all NLP subproblem using the solver CONOPT version 3.

- The GAMS option statement “OPTION SYSOUT = ON;” can generate extra solver information that can be helpful to diagnose problems.
- If many of the NLP subproblems are infeasible, add slacks as described in section 10.2.
- 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 7.1.
- If you have access to another MINLP solver such as SBB, try to use a different solver on your model. To select 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. It will write a model in scalar GAMS notation, which 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.

DICOPT References

[1] M. A. Duran and I. E. Grossmann, *An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs*, Mathematical Programming, 36 (1986), pp. 307–339.

[2] E. Kalvelagen, *Model building with GAMS*, to appear.

[3] G. R. Kocis and I. E. Grossmann, *Relaxation Strategy for the Structural Optimization of Process Flowsheets*, Industrial and Engineering Chemistry Research, 26 (1987), pp. 1869–1880.

[4] G. R. Kocis and I. E. Grossmann, *Computational Experience with DICOPT solving MINLP Problems in Process Systems Engineering*, Computers and Chemical Engineering, 13 (1989), pp. 307–315.

[5] J. Viswanathan and I. E. Grossmann, *A combined Penalty Function and Outer Approximation Method for MINLP Optimization*, Computers and Chemical Engineering, 14 (1990), pp. 769–782.

[6] H. P. Williams, *Model Building in Mathematical Programming*, 4-th edition (1999), Wiley.

EMP

Contents

1 Introduction

EMP (Extended Mathematical Programming) is not a solver but an (experimental) framework for automated mathematic 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, but additional features are being added regularly.

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 and 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

```
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 option

```
dict dict.txt
```

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.

MCF: Terminate and EMPFileName

2.2 The EMP Info File

MCF: Details on how to write this, etc. Maybe at end of document?

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:

```
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:

$$\min_x f(x) \text{ s.t. } g(x) \le 0,\ h(x) = 0. \tag{12.1}$$

the Lagrangian is:

$$\mathcal{L}(x,\lambda,\mu) = f(x) - \lambda^T g(x) - \mu^T h(x).$$

The first order conditions are the following MCP:

$$\begin{array}{lll} 0 = \nabla_x \mathcal{L}(x,\lambda,\mu) & \perp & x \text{ free} \\ 0 \ge -\nabla_\lambda \mathcal{L}(x,\lambda,\mu) & \perp & \lambda \le 0 \\ 0 = -\nabla_\mu \mathcal{L}(x,\lambda,\mu) & \perp & \mu \text{ free} \end{array}$$

A specific example is:

$$\begin{array}{ll} \min\limits_{x,y,z} & -3x + y \\ \text{s.t.} & x + y \le 1,\ x + y - z = 2,\ x, y \ge 0 \end{array}$$

which is found in the EMPLIB model nlp2mcp:

```
variables f,z; positive variables x,y;
equations g, h, defobj;

g.. x + y =l= 1;
h.. x + y - z =e= 2;
defobj.. f =e= -3*x + y;

model comp / defobj, g, h /;

file info / '%emp.info%' /;
putclose info / 'modeltype mcp';

solve comp using emp minimizing f;
```

Lines 10-11 write out the default “empinfo” file whose location is provided in the system string `%emp.info%`. Armed with this additional information, the EMP tool automatically creates the following MCP:

$$\begin{array}{lll} 0 \le -3 - \lambda - \mu & \perp & x \ge 0 \\ 0 \le 1 - \lambda - \mu & \perp & y \ge 0 \\ 0 = \mu & \perp & z \text{ free} \\ 0 \ge x + y - 1 & \perp & \lambda \le 0 \\ 0 = x + y - z - 2 & \perp & \mu \text{ free.} \end{array}$$

MCF: shouldn't keepobj (and objvarname) be an empinfo file directive? MCF: objvarname calls the objvar whatever you assign in the scalar model?

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

$$\begin{array}{rl} \min\limits_{x_1,x_2,x_3} & \exp(x_1) \\ \text{s.t.} & \log(x_1) = 1, \\ & x_2^2 \le 2, \\ & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \le 5, x_1 \ge 0, x_2 \ge 0. \end{array}$$

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.. sqr(x2) =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 x1.l = 1; x2.l = 1; x3.l = 1;
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:

$$\begin{aligned}\min_{x_1,x_2,x_3}\ & \exp(x_1) + 5\left\|\log(x_1) - 1\right\|^2 + 2\max(x_2^2 - 2, 0)\\ \text{s.t. } & x_1/x_2 = \log(x_3),\\ & 3x_1 + x_2 \le 5, x_1 \ge 0, x_2 \ge 0.\end{aligned}$$

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:

$$\begin{aligned}\min_{x_1,x_2,x_3,w}\ & \exp(x_1) + 5\left(\log(x_1) - 1\right)^2 + 2w\\ \text{s.t. } & x_1/x_2 = \log(x_3),\\ & 3x_1 + x_2 \le 5, x_1 \ge 0, x_2 \ge 0\\ & w \ge x_2^2 - 2, w \ge 0\end{aligned}$$

and recover a standard form NLP.

The “empinfo” file:

```
modeltype NLP
adjustequ
f1 sqr 5
f2 maxz 2
```

coupled with replacing line 15 with

```
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 explicity with the nonsmoothness. The empinfo file is given by:

```
modeltype NLP
adjustequ
f1 sqr 5
f2 maxz 2
```

and this generates the following MCP:

$$\begin{array}{lll}
0 = \log(x_1) - 1 + y_1/10 & \perp & y_1 \text{ free,} \\
0 \leq x_2^2 - 2 & \perp & y_2 \geq 0 \\
0 = x_1/x_2 - \log(x_3) & \perp & y_3 \text{ free,} \\
0 \geq 3x_1 + x_2 - 5 & \perp & y_4 \leq 0 \\
0 \leq exp(x_1) - y_1/x_1 - y_3/x_2 - 3y_4 & \perp & x_1 \geq 0 \\
0 \leq -2y_2x_2 + x_1y_3/x_2^2 - y_4 & \perp & x_2 \geq 0 \\
0 = y_3/x_3 & \perp & x_3 \text{ free,}
\end{array}$$

where y represent the multipliers.

A complete description of the process to derive this MCP will be given later in Section 10.

5 Dual Problems

MCF: Should add ability for form the dual problem here

6 Bilevel Programs

Mathematical programs with optimization problems in their constraints have a long history in operations research including [?, ?, ?]. 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:

$$\begin{aligned}
&\min_{x,y} f(x,y) \\
&\text{s.t. } g(x,y) \leq 0, \\
&\qquad y \text{ solves } \min_{y} v(x,y) \text{ s.t. } \quad h(x,y) \geq 0.
\end{aligned}$$

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:

$$\begin{aligned}
&\min_{x,y} f(x,y) \\
&\text{s.t. } g(x,y) \leq 0, \\
&\qquad 0 = \nabla_y v(x,y) - \lambda^T \nabla_y h(x,y) \perp x \text{ free} \\
&\qquad 0 \leq h(x,y) \perp \lambda \geq 0.
\end{aligned}$$

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 [?]. Mathematically, the problem is

$$\begin{aligned} \min_{x,y}\ & x - 4y \\ \text{s.t. } & y \text{ solves } \min_{y}\ y \\ & \qquad\qquad \text{s.t. } x + y \geq 3 \\ & \qquad\qquad\qquad 2x - y \geq 0 \\ & \qquad\qquad\qquad -2x - y \geq -12 \\ & \qquad\qquad\qquad -3x + 2y \geq -4 \end{aligned}$$

and the EMPLIB model bard511 contains the following code:

```
1  positive variables x,y; variables objout,objin;
2  equations defout,defin,e1,e2,e3,e4;
3
4  defout.. objout =e= x - 4*y;
5  defin..  objin  =e= y;
6
7  e1..      x +   y =g=   3;
8  e2..    2*x -   y =g=   0;
9  e3..   -2*x -   y =g= -12;
10 e4..   -3*x + 2*y =g=  -4;
11
12 model bard / all /;
13
14 $echo bilevel x min objin y defin e1 e2 e3 e4 > "%emp.info%"
15
16 solve bard using emp minimizing objout;
```

Note that lines 1-12 define the functions that form the objectives and constraints of the model and assemble them into the model. Line 14 writes the “empinfo” file and states that the lower level problem involves the objective 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:

```
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:

```
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 explicity names all the variables in each followers problem as shown before.

7 Variational Inequalities

A variational inequality VI(F, X) is to find $x \in X$:

$$F(x)^T(z - x) \geq 0, \text{ for all } z \in X.$$

Here X is a closed (frequently assumed convex) set, defined for example as

$$X = \{x \mid x \geq 0, h(x) \geq 0\}. \tag{12.2}$$

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 (12.2) by introducing multipliers λ on the constraints h:

$$\begin{array}{lcl} 0 \leq F(x) - \lambda^T \nabla h(x) & \perp & x \geq 0 \\ 0 \leq h(x) & \perp & \lambda \geq 0. \end{array}$$

If X has a different representation, this construction would be modified appropriately.

A simple two dimensional example may be useful to improve understanding. Let

$$F(x) = \begin{bmatrix} x_1 + 2 \\ x_1 + x_2 - 3 \end{bmatrix}, \; X = \{x \geq 0 \mid x_1 + x_2 \leq 1\},$$

so that F is an affine function, but F is not the gradient of any function $f : \mathbf{R}^2 \to \mathbf{R}$. For this particular data, VI(F, X) has a unique solution $x = (0, 1)$.

```
sets J  / 1, 2 /;
positive variable x(J)  'vars, perp to f(J)';

equations F(J), h;

F(J)..  (x('1') + 2)$sameas(J,'1') + (x('1') + x('2') - 3)$sameas(J,'2') =n= 0 ;
h..     x('1') + x('2') =l= 1;

model simpleVI / F, h/;

file fx /"%emp.info%"/;
putclose fx 'vifunc F x h';

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 (vifunc) that pairs F with x and that the remaining (unpaired) equations form the constraint set X. 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 'vifunc F x';
```

In this case, by default any equations that are given in the model statement but not included as a pair in the vifunc statement are automatically used to form X. An alternative way to write this model without using "sameas" is given in EMPLIB as affinevi.

Further example models in EMPLIB are named: simplevi, simplevi2, simplevi3, target, traffic, traffic2, transvi and zerofunc.

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 multmpec.

8 Embedded Complementarity Systems

MCF: Need to get our names straight: embedded comp system vs equilibrium model

A different type of embedded optimization model that arises frequently in applications is:

$$\begin{array}{lll} \max\limits_{x} & f(x,y) & \\ \text{s.t.} & g(x,y) \leq 0 & (\perp p \geq 0) \\ & & \\ H(x,y,p) = 0 & & (\perp y \text{ free}) \end{array}$$

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.

MCF: Maybe use ferris43 model instead since it does everything. But simpequil2 should go into emplib.

A specific example is:

$$\begin{array}{lll} \max\limits_{x} & x & \\ \text{s.t.} & x + y \leq 1 & \\ & & \\ -3x + y = 0.5 & & (\perp y \text{ free}) \end{array}$$

which is found in the EMPLIB model simpequil2:

```
1  variables y; positive variables x;
2  equations optcons, vicons;
3
4  optcons.. x + y =l= 1;
5  vicons.. -3*x + y =e= 0.5;
6
7  model comp / optcons, vicons /;
8
9  file info / '%emp.info%' /;
10 put info / 'equilibrium';
11 put      / 'max x optcons';
12 putclose / 'vifunc vicons y';
13
14 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 1-7 above) in an "empinfo" file (lines 9-12). Specifically, line 10 indicates the problem is an equilibrium problem involving one or more agent problems. Line 11 defines the first agent as an optimizer (over x), and line 12 defines the second agent as solving a VI in y. Armed with this additional information, the EMP tool automatically creates the following MCP:

$$\begin{array}{lcl} 0 \le -1 + p & \perp & x \ge 0 \\ 0 \le 1 - x - y & \perp & p \ge 0 \\ 0 = -3x + y - 0.5 & \perp & y \text{ free,} \end{array}$$

(which is formed by the steps we outline below).

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 clasical PIES Model due to Hogan. In this setting, the problem is described by a linear program

$$\begin{array}{ll} \min_x & c^T x \\ \text{s.t.} & Ax = q(p) \\ & Bx = b \\ & x \ge 0 \end{array}$$

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
3 file myinfo /'%emp.info%'/;
4 put myinfo 'equilibrium ';
5 put 'min obj c o ct ot lt ht defobj dembal cmbal ombal lmbal hmbal
6 ruse ';
7 putclose 'dualvar p dembal';
8
9 solve piesemp using emp;
```

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 explicity claimed in line 5, along with all variables except 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. EMP explicitly enforces the rule that every variable and constrain

There are several shorthands possible here. The first is that line 5 can be replaced by the '*' form:

```
put 'min obj * defobj dembal cmbal ombal lmbal hmbal ruse ';
```

Alternatively, an even shorter version is possible since there is only one agent present in this model, namely:

```
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;

file myinfo /'%emp.info%'/;
putclose myinfo 'dualvar p dembal';

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:

```
min x optcons
vifunc vicons y
dualval p optcons
```

namely that the function H that is defined in vicons 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:

$$\begin{array}{lll} 0 = \nabla_x \mathcal{L}(x,y,p) & \perp & x \text{ free} \\ 0 \geq -\nabla_p \mathcal{L}(x,y,p) & \perp & p \leq 0 \\ 0 = H(x,y,p) & \perp & y \text{ free}, \end{array}$$

where the Lagrangian is defined as

$$\mathcal{L}(x,y,p) = f(x,y) - p^T g(x,y).$$

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 named: ferris43, flipper, pies, scarfemp-dual, simpequil, transecs, transeql

9 MOPECs

MCF: Use a Bimatrix game example?

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

$$x_i^* \in \arg\min_{x_i \in X_i} \ell_i(x_i, x_{-i}^*, q), \forall i \in \mathcal{I},$$

where x_{-i} are other players decisions and the quantities q are given exogenously, or via complementarity:

$$0 \leq H(x,q) \quad \perp \quad q \geq 0.$$

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 ith Nash player. Also there are the variables x_{-i} that are treated as parameters by the ith Nash player, but are treated as variables by a different player j.

While we do not specify the syntax here for these issues , EMPmanual 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 [**?**] 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.

10 Extended Nonlinear Programs

Optimization models have traditionally been of the form (12.1). 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 [**?**, **?**, **?**] have introduced the notion of extended nonlinear programming, where the (primal) problem has the form:

$$\min_{x\in X} f(x) + \theta(-g_1(x), \ldots, -g_m(x)). \tag{12.3}$$

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 θ can be thought of as a generalized penalty function that may well be nonsmooth. However, when θ has the following form

$$\theta(u) = \sup_{y\in Y}\{y^T u - k(y)\}, \tag{12.4}$$

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 θ 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 that 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 a re computationally more appealing; such reformulations can be generated using different options to JAMS.

10.1 Forms of θ

The EMP model type makes the problem format (12.3) 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 θ that can be generated by particular choices of Y and k. In many cases, the function θ is separable, that is

$$\theta(u) = \sum_{i=1}^{m} \theta_i(u_i).$$

so we can either specify θ_i or θ itself.

Extended nonlinear programs include the classical nonlinear programming form (12.1) as a special case. This follows from the observation that if K is a closed convex cone, and we let ψ_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 (12.1) can be rewritten as:

$$\min_x f(x) + \psi_K((-g(x), -h(x)), \; K = \mathbf{R}^m_+ \times \{0\}^p,$$

where m and p are the dimensions of g and h respectively and $\mathbf{R}^m_+ = \{u \in \mathbf{R}^m \mid u \geq 0\}$. An elementary calculation shows that

$$\psi_K(u) = \sup_{v \in K^\circ} u^T v,$$

where $K^\circ = \left\{u \mid u^T v \leq 0, \forall v \in K\right\}$ is the polar cone of the given cone K. Thus, when $\theta(u) = \psi_K(u)$ we simply take

$$k \equiv 0 \text{ and } Y = K^\circ. \tag{12.5}$$

In our example, $K^\circ = \mathbf{R}^m_- \times \mathbf{R}^p$. 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 θ: Formally, for $u \in \mathbf{R}$,

$$\theta(u) = \begin{cases} \rho u & \text{if } u \geq 0 \\ \sigma u & \text{else.} \end{cases}$$

In this case, simple calculations prove that θ has the form (12.4) for the choices:

$$k \equiv 0 \text{ and } Y = [\sigma, \rho].$$

The special case where $\sigma = -\rho$ results in

$$\theta(u) = \rho \left| u \right|. \tag{12.6}$$

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 ∞ 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

$$\theta(u) = \gamma u^2 \tag{12.7}$$

then a simple calculation shows that we should take

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = \mathbf{R}.$$

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 (12.6) and (12.7) 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 θ continuously differentiable - see [**?**]. That is:

$$\text{i.e. } \theta(u) = \begin{cases} \gamma u - \frac{1}{2}\gamma^2 & \text{if } u \geq \gamma \\ \frac{1}{2}u^2 & \text{if } u \in [-\gamma, \gamma] \\ -\gamma u - \frac{1}{2}\gamma^2 & \text{else.} \end{cases}$$

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

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = [\sigma, \rho],$$

with $\alpha = \frac{\sigma}{2\gamma}$ and $\beta = \frac{\rho}{2\gamma}$.

The final example that we give is that of L_∞ penalization. This example is different to the examples given above in that θ 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

$$k \equiv 0 \text{ and } Y = \left\{ y \in \mathbf{R}^m \mid y \geq 0, \sum_{i=1}^m y_i = 1 \right\},$$

that is, Y is the unit simplex.

10.2 Underlying theory

The underlying structure of θ 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 θ function. The theory is an elegant combination of calculus arguments related to g_i and its derivatives, and variational analysis for features related to θ.

It is shown in [?] that under a standard constraint qualification, the first-order conditions of (12.3) are precisely in the form of the following variational inequality:

$$\text{VI}\left(\begin{bmatrix} \nabla_x \mathcal{L}(x,y) \\ -\nabla_y \mathcal{L}(x,y) \end{bmatrix}, X \times Y \right). \tag{12.8}$$

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 (12.3) is given to EMP, then the optimality conditions (12.8) 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 of k and Y given in (12.5), then the optimality conditions of (12.3) are precisely the standard complementarity problem given as (**??**). 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 (12.8) is a saddle point for the Lagrangian on $X \times Y$. Furthermore, in this setting, the saddle point generates solution s to the primal problem (12.3) and its dual problem:

$$\max_{y \in Y} d(y), \quad \text{where } d(y) = \inf_{x \in X} \mathcal{L}(x,y),$$

with no duality gap.

10.3 A simple example

MCF: This repeats stuff from earlier section, needs work. EMPLIB model is simpenlp

As an example, consider the problem

$$\begin{aligned} \min_{x_1,x_2,x_3} \quad & \exp(x_1) + 5\left\|\log(x_1) - 1\right\|^2 + 2\max(x_2^2 - 2, 0) \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

In this problem, we would take

$$X = \left\{x \in \mathbf{R}^3 \mid 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0\right\}.$$

The function θ essentially treats 3 separable pieces:

$$\begin{aligned} g_1(x) &= \log(x_1) - 1, \\ g_2(x) &= x_2^2 - 2, \\ g_3(x) &= x_1/x_2 - \log(x_3). \end{aligned}$$

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:

$$\begin{aligned} \min_{x_1,x_2,x_3,w} \quad & \exp(x_1) + 5\left\|\log(x_1) - 1\right\|^2 + 2w \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \\ & w \geq x_2^2 - 2, w \geq 0 \end{aligned}$$

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

$$\begin{aligned} \theta_1(u_1) &= 5u_1^2, \\ \theta_2(u_2) &= 2\max(u_2, 0), \\ \theta_3(u_3) &= \psi_{\{0\}}(u_3). \end{aligned}$$

The discussion above allows us to see that

$$\begin{aligned} Y &= \mathbf{R} \times [0,2] \times \mathbf{R}, \\ k(y) &= \frac{1}{20}y_1^2 + 0 + 0. \end{aligned}$$

The corresponding Lagrangian is the smooth function:

$$\mathcal{L}(x,y) = f(x) - \sum_{i=1}^{3} y_i g_i(x) - k(y).$$

The corresponding VI (12.8) 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

$$g_4(x) = 3x_1 + x_2 - 5,\ \theta_4(u) = \psi_{\mathbf{R}_+}$$

resulting in the following choices for Y and k:

$$\begin{aligned} Y &= \mathbf{R} \times [0,2] \times \mathbf{R} \times \mathbf{R}_-, \\ k(y) &= \frac{1}{20}y_1^2 + 0 + 0 + 0. \end{aligned}$$

Since X and Y are now simple bound sets, (12.8) 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
```

The full model and option files are available in [?].

10.4 Reformulation as a classical NLP

Suppose

$$\theta(u) = \sup_{y \in Y}\{u^T y - \frac{1}{2}y^T Q y,\}$$

for a polyhedral set $Y \in \mathbf{R}^m$ and a symmetric positive semidefinite $Q \in \mathbf{R}^{m\times m}$ (possibly $Q = 0$). Suppose further that

$$\begin{aligned} &X = \{x \mid Rx \le r\},\ \ Y = \left\{y \mid S^T y \le s\right\}, \\ &Q = DJ^{-1}D^T,\ \ F(x) = (g_1(x), \dots, g_m(x)), \end{aligned}$$

where J is symmetric and positive definite (for instance $J = I$). Then, as outlined by [?], the optimal solutions $\bar{x}$ of (12.3) are the $\bar{x}$ components of the optimal solutions $(\bar{x}, \bar{z}, \bar{w})$ to

$$\begin{array}{ll} \min & f(x) + s^T z + \frac{1}{2}w^T J w \\ \text{s.t.} & Rx \le r, z \ge 0, F(x) - Sz - Dw = 0. \end{array}$$

The multiplier on the equality constraint in the usual sense is the multiplier associated with $\bar{x}$ in the extended Lagrangian for (12.3). (Note that a Cholesky factorization may be needed to determine D.)

It may be better to solve this reformulated NLP than to solve (12.8). 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.

Example models are named:

11 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 [?] 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 comes before job j or the converse. Such a disjunction can be specified using an empinfo file containing lines:

```
  disjuncton * 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
```

would add binary variables and constraints to impose the disjunction using a Big-M value of 1000. Alternatively, for the CPLEX solver, the option setting (for EMP):

```
  default indic
```

writes out a model and a CPLEX option file that implements a reformulation using indicator constraints. The EMP model library that is part of the standard GAMS distribution contains a 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 [?] and a synthesis problem involving 8 processes from [?] are also included in the EMP model library. As a final example, the gasoline emission model outlined in [?] is precisely in the form that could exploit the features of EMP related to (nonlinear) disjunctive programming.

Example models are named: makespan, sequence.

12 Options available

The empinfo file has a vectorized format and a more powerful (but more complex) scalar version.

The JAMS solver has the following options:

Can pass user defined parameters to the subsolver ("subsolverpar").

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 {equ|*}

AdjustEqu equ abs|sqr|maxz|huber|... {weight {param}}

ModelType MCP|NLP|MIP|...

BiLevel {var} {MAX|MIN obj {var|*} {[-] equ}} {VI {var|*} {[-] equ var} {[-] equ}} {DualVar {var [-] equ}}

Equilibrium {MAX|MIN obj {var|*} {[-] equ}} {VI {var|*} {[-] equ var} {[-] equ}} {DualVar {var [-] equ}}

VI {var|*} {[-] equ var} {[-] equ}

DualEqu {[-] equ var}

DualVar {var [-] equ}

-------
[ ] optional      | exclusive      { } can be repeated
```

Last modified February 12, 2011 by MCF

EXAMINER

Contents

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. It 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 (indeed, perhaps all of them) are already being done by the GAMS solvers, so Examiner is in a sense redundant. However, a facility 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 it does not appear as an option in the list of possible solver defaults when installing GAMS. However, you can choose Examiner via the command line:

```
gams trnsport 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, and this 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 2.4). 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, instead of using the values passed in by GAMS for these.

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 & 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 defined in Section 2.1. 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 the 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 3.2).

There exist different ways to check the items mentioned above. For example, when checking for primal feasibility, different norms can be used to measure the error of the residual. Currently, we have only implemented one way to make these checks. TODO: document this implementation, perhaps add others.

2.3 Scaling

By default, Examiner makes its checks on the original, unscaled model. In many cases, though, 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 construct precisely the same scaled model, we can expect to get a better idea of how the solver did by looking at a model scaled by Examiner 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

$$\max(\|RHS_i\|, \max_j(\|A_{ij}\| \cdot \max(1, \|x_j\|)))$$

In this way variables with a large level value lead to large scale factors. To make the scale factor independent of the variable values, use an option file line of “AbsXScale 0”. 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

For details on how to create and use an option file, see the introductory chapter on solver usage.

3.1 General Options

The following general 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.

Option	Description	Default
`absxscale`	If on, the matrix coefficients are multiplied by max(1,abs(x)) when computing the scale factors. If off, the matrix coefficients are taken as is. See Section 2.3	on
`dumpgamspoint`	If on, dump the `gamspoint` to a basis file in GAMS source format.	off
`dumpinitpoint`	If on, dump the `initpoint` to a basis file in GAMS source format.	off
`dumpsolupoint`	If on, dump the `solupoint` to a basis file in GAMS source format.	off
`dumpsolvpoint`	If on, dump the `solvpoint` to a basis file in GAMS source format.	off
`examinegamspoint`	If on, examine the `gamspoint`.	off
`examineinitpoint`	If on, examine the `initpoint`. By default, this option is on if GAMS/Base passes an advanced basis, and off otherwise.	auto
`examinesolupoint`	If on, examine the `solupoint`. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
`examinesolvpoint`	If on, examine the `solvpoint`. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
`fcheckall`	If set, this option forces all the checks from Section 2.2 on or off.	auto
`fcheckdcon`	If set, this option forces the dual contraint feasibility check from Section 2.2 on or off.	auto
`fcheckdcmp`	If set, this option forces the dual complementary slackness check from Section 2.2 on or off.	auto
`fcheckdvar`	If set, this option forces the dual variable feasibility check from Section 2.2 on or off.	auto
`fcheckpcon`	If set, this option forces the primal constraint feasibility check from Section 2.2 on or off.	auto
`fcheckpcmp`	If set, this option forces the primal complementary slackness check from Section 2.2 on or off.	auto
`fcheckpvar`	If set, this option forces the primal variable feasibility check from Section 2.2 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.	off
`returngamspoint`	If on, return the `gamspoint` as a solution to GAMS/Base.	off
`returninitpoint`	If on, return the `initpoint` as a solution to GAMS/Base.	auto
`returnsolupoint`	If on, return the `solupoint` as a solution to GAMS/Base.	auto
`returnsolvpoint`	If on, return the `solvpoint` as a solution to GAMS/Base.	auto
`scaled`	If set, examiner checks will be made on the scaled model.	no
`scaleLB`	Lower bound for applied row scales.	1.0
`scaleUB`	Upper bound for applied row scales.	1.0e300
`subsolver`	Indicates what subsolver to run. By default, the subsolver used is the default subsolver for the model type in question.	auto
`subsolveropt`	If set, indicates what optfile value to pass to the subsolver used. Can also be set via the `subsolver` option by appending a `.n` to the subsolver name, e.g. `subsolver bdmlp.3`	auto
`trace`	If set, trace information will be computed and appended to this file. By	none

3.2 Tolerance Options

The following options can be used to set the various tolerances to non-default values.

Option	Description	Default
`dualcstol`	Dual complementary slackness tolerance. By dual CS we refer to complementary slackness between the dual variables and the primal constraints.	1e-7
`dualfeastol`	Dual feasibility tolerance. This tolerance is used for the checks on the dual variables and the dual constraints.	1e-6
`ectol`	Equilibrium condition complementarity tolerance. Applicable to MCP and MPEC models, where the equilibrium conditions are given by the equation-variable pairs in the model statement.	1e-6
`primalcstol`	Primal complementary slackness tolerance. By primal CS we refer to complementary slackness between the primal variables and the dual constraints.	1e-7
`primalfeastol`	Primal feasibility tolerance. This tolerance is used for the checks on the primal variables and the primal constraints.	1e-6

GAMS/AMPL

Contents

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 *trnsport.gms* model, we would run

```
>> gams trnsport.gms lp=ampl
```

As 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.

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. Finally if `amplpath.txt` is not found, the GAMS will try the system path.

If no AMPL exectuable is found, the user will see a message similar to

```
AMPL Link 0. Jan 26, 2005 LNX.00.NA 21.6 002.000.000.LXI P3PC
```

```
--- No AmplPath option or "amplpath.txt" file found
--- System PATH will be used
```

There may also be an output indicating that AMPL was not found, either because it is not installed or because it is not found in the system path.

3 GAMS/AMPL Options

GAMS/AMPL solver options are passed on through solver option files. If you specify "`<modelname>.optfile = 1;`" before the SOLVE statement in your GAMS model, GAMS/AMPL will then look for and read an option file with the name *ampl.opt* (see "Using Solver Specific Options" for general use of solver option files).

Option	**Description**	**Default**
`AmplPath`	Path to AMPL system files	
`DotMod`	Ampl input file name	`ampl.mod`
`Help`	Display GAMS/AMPL options	
`Option`	Verbation AMPL options	
`RunAmpl`	Name of AMPL executable	
`TolNone`	Tolerance to interpret status none	`1e-12`

The `Option` specifier is used to specify AMPL options as 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

```
option minos_options "timing=3 outlev=2";
```

GAMS/LINGO

Contents

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 *trnsport.gms* model, we would run

```
>> gams trnsport.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 exectuable 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).

Option	**Description**	**Default**
`DotLng`	Lingo input file name	`lingo.lng`
`Help`	Display GAMS/LINGO options	
`LingoPath`	Path to LINGO system files	
`OptCR`	Relative termination for MIPs and Global nonlinear optimization problems	
`IterLim`	Minor iteration limit.	
`ResLim`	Resource limit.	

GUROBI 4.5

Gurobi Optimization, `www.gurobi.com`

Contents

1 Introduction

The Gurobi suite of optimization products include state-of-the-art simplex and parallel barrier based linear programming (LP) and quadratic programming (QP) solvers, as well as parallel mixed-integer linear programming (MILP) and mixed-integer quadratic programming (MIQP) solvers.

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.

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 and Quadratic Programming

Gurobi can solve LP and QP problems using several alternative algorithms. The majority of LP or QP problems solve best using Gurobi's state-of-the-art dual simplex algorithm. Certain types of problems benefit from using the parallel barrier or the primal simplex algorithms. 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.

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. Often the solve from scratch of a presolved model outperforms a solve from an unpresolved model started from an advanced basis. It is impossible to determine a priori if presolve or starting from a given advanced basis without presolve will be faster. By default, GAMS/Gurobi 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 GAMS/Gurobi option usebasis can be used to ignore a basis 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 LP or QP 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/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.

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

$$|BP - BF| < |BF| * \text{OptCR}$$

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:

$$|BP - BF| < |BP| * \text{OptCR}$$

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`.

5 Summary of GUROBI Options

5.1 Termination options

bariterlimit	Limits the number of barrier iterations performed
cutoff	Sets a target objective value

iterationlimit	Limits the number of simplex iterations performed
nodelimit	Limits the number of MIP nodes explored
solutionlimit	Limits the number of feasible solutions found
timelimit	Limits the total time expended in seconds

5.2 Tolerance options

barconvtol	Controls barrier termination
feasibilitytol	Primal feasibility tolerance
intfeastol	Integer feasibility tolerance
markowitztol	Threshold pivoting tolerance
mipgap	Relative MIP optimality gap
mipgapabs	Absolute MIP optimality gap
optimalitytol	Dual feasibility tolerance
psdtol	limit on the amount of diagonal perturbation

5.3 Simplex and Barrier options

barcorrectors	Limits the number of central corrections performed in each barrier iteration
barorder	Chooses the barrier sparse matrix fill-reducing algorithm
crossover	Determines the crossover strategy used to transform the barrier solution into a basic solution
crossoverbasis	Determines the initial basis construction strategy for crossover
normadjust	Pricing norm variants
objscale	Objective coefficients scaling
perturbvalue	Magnitude of simplex perturbation when required
quad	Quad precision computation in simplex
scaleflag	Enables or disables model scaling
simplexpricing	Determines variable pricing strategy

5.4 MIP options

branchdir	Determines which child node is explored first in the branch-and-cut search
cliquecuts	Controls clique cut generation
covercuts	Controls cover cut generation
cutaggpasses	Maximum number of aggregation passes during cut generation
cutpasses	Maximum number of cutting plane passes performed during root cut generation
cuts	Global cut generation control
flowcovercuts	Controls flow cover cut generation
flowpathcuts	Controls flow path cut generation
gomorypasses	Maximum number of Gomory cut passes
gubcovercuts	Controls GUB cover cut generation
heuristics	Controls the amount of time spent in MIP heuristics
impliedcuts	Controls implied bound cut generation
improvestartgap	Optimality gap at which the MIP solver resets a few MIP parameters
improvestarttime	Elapsed time after which the MIP solver resets a few MIP parameters
minrelnodes	Number of nodes to explore in the Minimum Relaxation heuristic
mipfocus	Controls the focus of the MIP solver
mipsepcuts	Controls MIP separation cut generation

mircuts	Controls MIR cut generation
modkcuts	Controls the generation of mod-k cuts
networkcuts	Controls network cut generation
nodefiledir	Nodefile directory
nodefilestart	Nodefile starting indicator
nodemethod	Algorithm used to solve node relaxations in a MIP model
pumppasses	Number of passes of the feasibility pump heuristic
rins	Frequency of the RINS heuristic
submipcuts	Controls the generation of sub-MIP cutting planes
submipnodes	Limits the number of nodes explored by the heuristics
symmetry	Controls MIP symmetry detection
varbranch	Controls the branch variable selection strategy
zerohalfcuts	Controls zero-half cut generation

5.5 Other options

aggregate	Enables or disables aggregation in presolve
aggfill	Controls the amount of fill allowed during presolve aggregation
displayinterval	Controls the frequency at which log lines are printed in seconds
dumpsolution	Controls export of alternate MIP solutions
fixoptfile	Option file for fixed problem optimization
iis	Run the IIS finder if the problem is infeasible
iismethod	Controls use of IIS method
kappa	Display condition number of the basis matrix
method	LP and QP algorithm
mipstart	Use mip starting values
names	Indicator for loading names
precrush	Presolve constraint option
predual	Controls whether presolve forms the dual of a continuous model
predeprow	Controls the presolve dependent row reduction
premiqpmethod	Transformation presolve performs on MIQP models
prepasses	Controls the number of passes performed by presolve
presolve	Controls the presolve level
printoptions	List values of all options to GAMS listing file
readparams	Read Gurobi parameter file
rerun	Resolve without presolve in case of unbounded or infeasible
sensitivity	Provide sensitivity information
solvefixed	Indicator for solving the fixed problem for a MIP to get a dual solution
threads	Controls the number of threads to apply to parallel MIP or Barrier
usebasis	Use basis from GAMS
writeparams	Write Gurobi parameter file
writeprob	Save the problem instance

5.6 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   1.200000e+31   1.485042e-04      0s
    5624    1.1486966e+05   0.000000e+00   0.000000e+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 infeasibilites 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)

                  Objective                Residual
```

```
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   1.11502515e+13 -3.03102251e+08  7.65e+05 9.29e+07  2.68e+09     2s
   1   4.40523949e+12 -8.22101865e+09  3.10e+05 4.82e+07  1.15e+09     3s
   2   1.18016996e+12 -2.25095257e+10  7.39e+04 1.15e+07  3.37e+08     4s
   3   2.24969338e+11 -2.09167762e+10  1.01e+04 2.16e+06  5.51e+07     5s
   4   4.63336675e+10 -1.44308755e+10  8.13e+02 4.30e+05  9.09e+06     6s
   5   1.25266057e+10 -4.06364070e+09  1.52e+02 8.13e+04  2.21e+06     7s
   6   1.53128732e+09 -1.27023188e+09  9.52e+00 1.61e+04  3.23e+05     9s
   7   5.70973983e+08 -8.11694302e+08  2.10e+00 5.99e+03  1.53e+05    10s
   8   2.91659869e+08 -4.77256823e+08  5.89e-01 5.96e-08  8.36e+04    11s
   9   1.22358325e+08 -1.30263121e+08  6.09e-02 7.36e-07  2.73e+04    12s
  10   6.47115867e+07 -4.50505785e+07  1.96e-02 1.43e-06  1.18e+04    13s
  ......
  26   1.12663966e+07  1.12663950e+07  1.85e-07 2.82e-06  1.74e-04     2s
  27   1.12663961e+07  1.12663960e+07  3.87e-08 2.02e-07  8.46e-06     2s

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                2s
       0 DPushes remaining with DInf 2.8167333e-06                2s

     180 PPushes remaining with PInf 0.0000000e+00                2s
       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   0.000000e+00   0.000000e+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              45          20.0000    26.0300  30.2%  28.4    0s
*  353    71              29          21.0000    25.0000  19.0%  19.3    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

aggregate (*integer*) Enables or disables aggregation in presolve

(default = 1)

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)

bariterlimit (*integer*) Limits the number of barrier iterations performed

(default = infinity)

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)

barorder (*integer*) Chooses the barrier sparse matrix fill-reducing algorithm

(default = -1)

-1	Auto
0	Approximate Minimum Degree ordering
1	Nested Dissection ordering

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

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)

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 generation. 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

displayinterval *(integer)* Controls the frequency at which log lines are printed in seconds

(default = 5)

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)

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)

-1 Auto

0 Off

1 Conservative

2 Aggressive

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 IIS finder if the problem is infeasible

(default = 0)

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)

-1 Auto

0 Off

1 Conservative

2 Aggressive

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)

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 *IntFeasTol* from the nearest integer value.

Range: [1e-9,1e-1]

(default = 1e-5)

iterationlimit (*real*) Limits the number of simplex iterations performed

(default = infinity)

kappa (*integer*) Display condition number of the basis matrix

(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*) LP and QP algorithm

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.

(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,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)

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)

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

names *(integer)* Indicator for loading names

(default = 1)

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

normadjust ***(integer)*** Pricing norm variants

Chooses from among multiple pricing norm variants. The default value of -1 chooses automatically.

(default = -1)

objscale ***(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)

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)

predeprow ***(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)

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)

presolve *(integer)* Controls the presolve level

(default = -1)

-1	Auto
0	Off
1	Conservative
2	Aggressive

printoptions *(integer)* List values of all options to GAMS listing file

(default = 0)

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,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)

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

(default = 1)

sensitivity *(integer)* Provide sensitivity information

(default = 0)

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)

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
0	Off
1	Conservative
2	Aggressive

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.

(default = GAMS threads)

timelimit *(real)* Limits the total time expended in seconds

(default = GAMS reslim)

usebasis *(integer)* Use basis from GAMS

(default = 0)

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

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

Gather-Update-Solve-Scatter (GUSS)

1 Introduction

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:

$$\min_{x \in X(s)} f(x;s) \text{ s.t. } g(x;s) \leq 0 \tag{17.1}$$

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 as given above is simply for concreteness; equality constraints, range and bound constraints are trivial extensions of the above framework. Clearly the problems $\mathcal{P}(s)$ are interlinked and 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 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 back the solution 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:

$$\min c^T x \text{ s.t. } Ax \geq b,\ \ell \leq x \leq u.$$

The data in this problem is (A, b, c, ℓ, 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 ℓ:

```
1  Set i / ... /, j / ... /;
2  Parameter
3      A(i,j), b(i);
4  Variable
5      x(j), z, ...;
6  Equation
7      e(i), ...;
8  e(i).. sum(j, A(i,j)*x(j)) =g= b(i);
9  ...
10 model mymodel /all/;
11
12 Set s / s1*s10 /
13 Parameter
14     A_s(s,i,j) Scenario data
15     xlo_s(s,j) Scenario lower bound for variable x
16     xl_s(s,j)  Scenario solution for x.l
17     em_s(s,i)  Scenario solution for e.m;
18 Loop(s,
19   A(i,j) = A_s(s,i,j);
20   x.lo(j)= xlo_s(s,j);
21   solve mymodel min z using lp;
22   xl_s(s,j) = x.l(j);
23   em_s(s,i) = e.m(i);
24 );
```

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 manager or short GUSS is to provide syntax at the GAMS modeling level that makes an instance of a problem and allows the modeler limited access to treat that instance as an object, and 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 18-24 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`:

Input:

`param`:	Supplies scenario data for a parameter used in the model
`lower`:	Supplies scenario lower bounds for a variable
`upper`:	Supplies scenario upper bounds for a variable
`fixed`:	Supplies scenario fixed bounds for a variable

Output:

`level`:	Stores the levels of a scenario solution of variable or equation
`marginal`:	Stores the marginals of a scenario solution of variable or equation

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 with the model instance all the necessary expressions of the constraints, 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)*f*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)*f`, but with GUSS it is sufficient to supply different values for `f` that potentially result in many modified coefficient on the matrix level. The evaluation of the shipping cost term and the communication of the resulting matrix coefficient to the solver are done reliably behind the scenes by GUSS.

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. So in such a case, GUSS only communicates the changes from the previous model instance to the solver. This not only reduces the amount of data communicated to the solver, but 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.

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:

`OptfileInit`:	Option file number for the first solve
`Optfile`:	Option file number for subsequent solves
`LogOption`:	Determines amount of log output: `0` - Moderate log (default) `1` - Minimal log `2` - Detailed log
`SkipBaseCase`:	Switch for solving the base case (0 solves the base case)
`UpdateType`:	Scenario update mechanism: `0` - Set everything to zero and apply changes (default) `1` - Reestablish base case and apply changes `2` - Build on top of last scenario and apply changes
`RestartType`:	Determines restart point for the scenarios `0` - Restart from last solution (default) `1` - Restart from solution of base case `2` - Restart from input point

For the example model above the `UpdateType` setting would mean:

```
UpdateType=0:  loop(s, A(i,j) = A_s(s,i,j))
UpdateType=1:  loop(s, A(i,j) $= A_s(s,i,j))
UpdateType=2:  loop(s, A(i,j) = A_base(i,j);
                       A(i,j) $= A_s(s,i,j))
```

The option `SkipBaseCase=1` allows 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 status 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. The following solution status labels can be reported:

```
domusd    iterusd  objest     nodusd     modelstat  numnopt
numinfes  objval   rescalc    resderiv   resin      resout
resusd    robj     solvestat  suminfes
```

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 / modelstat, solvestat, objval /;
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.    xlo_s
           x.   level.    xl_s
           e.   marginal. em_s   /;
solve mymodel min z using lp scenario dict;
```

4 Implementation Details

This section describes some technical details that may provide useful insight in case of unexpected behavior.

GUSS changes all model parameters mentioned in the `dict` set to variables. So a linear model can produce some non-linear instructions (e.g. `d(i,j)*f*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 cannot be used that way any more. For example, suppose the model contains a constraint `e(i).. sum(j$A(i,j), ...)`. If `A(i,j)` is a parameter in the regular model, the compiler will not complain, but if `A` becomes a parameter that shows up in the first position of a `param` tuple in the `dict` set, the GAMS compiler will turn `A` into a variable and complain that an endogenous variable cannot be used in a `$`-condition.

The sparsity pattern of a model can be greatly effected 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 to delay the commitment for significant and permanent syntax changes of a developing method to handle model instances at a GAMS language level.

KNITRO

Contents

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

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) && (1.1a)\\ \text{subject to} \quad & c^L \le c(x) \le c^U && (1.1b)\\ & b^L \le x \le b^U, && (1.1c) \end{aligned}$$

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 μ. 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 (1.1) 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 8.

For a detailed description of the algorithm implemented in `Interior/CG` see [4] and for the global convergence theory see [1]. The method implemented in `Interior/Direct` is described in [8]. The `Active` algorithm is described in [3] and the global convergence theory for this algorithm is in [2]. 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 0 "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.

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.

***ModelName*.optCA = x;**

Absolute gap stop criterion for a discrete problem. The KNITRO option mip_integral_gap_abs takes its default from this value.

***ModelName*.optCR = x;**

Relative gap stop criterion for a discrete problem. The KNITRO option mip_integral_gap_rel takes its default from this value.

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:

```
# KNITRO-GAMS Options file
maxit        500
```

4.1 General Options

algorithm	controls which NLP algorithm to use
delta	initial trust region radius scaling factor
feastol	controls the relative feasibility tolerance
feastolabs	controls the absolute feasibility tolerance
gradopt	controls how to compute gradients
hessopt	controls how to compute Hessians
honorbnds	controls satisfaction of variable bounds
linsolver	controls which linear system solver to use
maxcgit	controls CG iteration limit
maxcrossit	controls crossover iteration limit
maxit	controls iteration limit

maxtime_cpu	CPU time limit
maxtime_real	real or wall-clock time limit
objrange	controls unboundedness check limit
opttol	controls the relative optimality tolerance
opttolabs	controls the absolute optimality tolerance
outlev	controls the output level
pivot	controls the initial pivot threshold
scale	controls scaling of model
soc	controls second order correction steps
xtol	controls termination based on stepsize

4.2 Barrier Options

bar_feasible	control for entering feasible mode
bar_feasmodetol	feasible mode tolerance
bar_initmu	control initial barrier parameter value
bar_initpt	control initial point strategy
bar_murule	control barrier parameter update strategy

4.3 MINLP Options

NOTE: the KNITRO library uses the `mip_` prefix for options and calls that are specific to problems with binary or integer variable, including mixed-integer nonlinear problems. This is in constrast to the GAMS convention, where `MIP` denotes a mixed-integer linear program and `MINLP` denotes a mixed-integer nonlinear program. We use the KNITRO convention to avoid changing the option names.

mip_branchrule	to use for MIP B&B
mip_gub_branch	toggles branching on generalized upper bounds
mip_heuristic	used in searching for an initial integer feasible point
mip_heuristic_maxit	iteration limit for heuristic
mip_implications	toggles addition of derived constraints
mip_integer_tol	integrality tolerance for discrete vars
mip_integral_gap_abs	absolute gap stop tolerance
mip_integral_gap_rel	relative gap stop tolerance
mip_lpalg	LP subsolver to use
mip_maxnodes	limit number of nodes explored
mip_maxsolves	limit subproblem solves allowed
mip_maxtime_cpu	cumulative CPU time limit
mip_maxtime_real	cumulative real or wall-clock time limit
mip_method	controls method to use
mip_outinterval	controls output frequency
mip_outlevel	controls output level
mip_rootalg	controls algorithm used for root node solve
mip_rounding	controls which rounding rule to apply
mip_selectrule	controls node selection rule
mip_strong_candlim	candidate limit for strong branching
mip_strong_level	controls tree levels for strong branching
mip_strong_maxit	iteration limit for strong branching
mip_terminate	controls termination test

4.4 Multi-Start Options

ms_enable	toggles the multi-start method
ms_maxbndrange	Maximum range to vary unbounded x when generating start points
ms_maxsolves	Specifies the maximum number of start points to try
ms_maxtime_cpu	cumulative CPU time limit
ms_maxtime_real	cumulative real or wall-clock time limit
ms_startptrange	Maximum range to vary all x when generating start points
ms_terminate	controls termination test

5 Detailed Descriptions of Knitro Options

algorithm *(integer)*

Controls which NLP algorithm to use.

0 Knitro will automatically try to choose the best algorithm based on the problem characteristics

1 Knitro will use the Interior/Direct algorithm

2 Knitro will use the Interior/CG algorithm

3 Knitro will use the Active Set algorithm

(default = 0)

bar_feasible *(integer)*

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 `algorithm=2` or `3`. See section 9.2 for more details.

0 No special emphasis on feasibility.

1 Iterates must satisfy inequality constraints 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.

Options 1 and 3 above activate the feasible version of Knitro. Given an initial point which *sufficiently* satisfies all *inequality* constraints as defined by,

$$cl + tol \leq c(x) \leq cu - tol \tag{5.2}$$

(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 (5.2) for determining when the feasible mode is active is determined by the double precision parameter `bar_feasmodetol` described below. This tolerance (i.e. `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 (5.2)) 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)

bar_feasmodetol *(double)*

Specifies the tolerance in (5.2) 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 *feasible*=1.

(default = 1e-4)

bar_initmu (*double*)

Specifies the initial value for the barrier parameter μ.

(default = 1e-1)

bar_initpt (*integer*)

Indicates whether an initial point strategy is used.

0 KNITRO will automatically choose the initial point strategy

1 Shift the initial point to improve barrier algorithm performance

3 Do not alter the initial point supplied by the user

(default = 0)

bar_murule (*integer*)

Controls the barrier parameter update strategy.

0 KNITRO will automatically choose the rule for updating the barrier parameter

1 KNITRO will monotonically decrease the barrier parameter

2 KNITRO uses an adaptive rule based on the complementarity gap to determine the value of the barrier parameter at every iteration

3 KNITRO uses a probing (affine-scaling) step to dynamically determine the barrier parameter value at each iteration

4 KNITRO uses a Mehrotra predictor-corrector type rule to determine the barrier parameter with safeguards on the corrector step

5 KNITRO uses a Mehrotra predictor-corrector type rule to determine the barrier parameter without safeguards on the corrector step

6 KNITRO minimizes a quality function at each iteration to determine the barrier parameter

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)

delta (*double*)

Specifies the initial trust region radius scaling factor used to determine the initial trust region size.

(default = 1)

feastol (*double*)

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.

1.0e-6

feastolabs (*double*)

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.

0.0

gradopt (*integer*)

Specifies how to compute the gradients of the objective and constraint functions.

1 exact gradients computed by GAMS

2 gradients computed by forward finite differences

3 gradients computed by central finite differences

(default = 1)

hessopt *(integer)*

Specifies how to compute the (approximate) Hessian of the Lagrangian.

1 exact Hessians computed by GAMS

2 KNITRO will compute a (dense) quasi-Newton BFGS Hessian

3 KNITRO will compute a (dense) quasi-Newton SR1 Hessian

4 KNITRO will compute Hessian-vector products using finite-differences

5 exact Hessian-vector products computed by GAMS

6 KNITRO will compute a limited-memory quasi-Newton BFGS Hessian

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=4` and `hessopt=5` are not available when `algorithm=1`. See section 9.1 for more detail on second derivative options.

(default = 1)

honorbnds *(integer)*

Indicates whether or not to enforce satisfaction of the simple bounds (1.1c) throughout the optimization (see section 9.3).

0 KNITRO does not enforce that the bounds on the variables are satisfied at intermediate iterates.

1 KNITRO enforces that the initial point and all subsequent solution estimates satisfy the bounds on the variables (1.1c).

2 KNITRO enforces that the initial point satisfies the bounds on the variables (1.1c).

(default = 0)

linsolver *(integer)*

Indicates which linear solver to use to solve linear systems arising in KNITRO algorithms.

0 **auto**: let KNITRO automatically choose the linear solver.

1 **internal**: not currently used; reserved for future use. Same as auto for now.

2 **hybrid**: use a hybrid approach where the solver chosen depends on the particular linear system which needs to be solved.

3 **QR**: use a dense QR method. This approach uses LAPACK QR routines. Since it uses a dense method, it is only efficient for small problems. It may often be the most efficient method for small problems with dense Jacobians or Hessian matrices.

4 **MA27**: use the HSL MA27 sparse symmetric indefinite solver.

5 **MA57**: use the HSL MA57 sparse symmetric indefinite solver.

(default = 0)

maxcgit (*integer)*

Specifies the maximum allowable number of inner conjugate gradient (CG) iterations per KNITRO minor iteration.

0 KNITRO automatically determines an upper bound on the number of allowable CG iterations based on the problem size.

n At most n CG iterations may be performed during one KNITRO minor iteration, where $n > 0$.

(default = 0)

maxcrossit (*integer)*

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)*

Specifies the maximum number of iterations before termination.

0 KNITRO 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 $n > 0$.

(default = 0)

maxtime_cpu (*double)*

Specifies the CPU time limit, in seconds.

(default = 1e8)

maxtime_real (*double)*

Specifies the real or wall-clock time limit, in seconds.

(default = 1e8)

mip_branchrule (*integer)*

Branching rule to use for MIP B&B.

0 automatic

1 use most fractional (most infeasible) branching

2 use pseudo-cost branching

3 use strong branching

(default = 0)

mip_gub_branch (*boolean)*

Toggles branching on generalized upper bounds.

(default = false)

mip_heuristic (*integer*)

Heuristic to use in searching for an initial integer feasible point.

0 automatic

1 none

2 feasibility pump

3 heuristic based on MPEC formulation

(default = 0)

mip_heuristic_maxit (*integer*)

Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled.

(default = 100)

mip_implications (*boolean*)

Toggles addition of constraints derived from logical implications.

(default = true)

mip_integer_tol (*double*)

Specifies the integrality tolerance for discrete variables.

(default = 1e-8)

mip_integral_gap_abs (*double*)

The absolute integrality gap stop tolerance. If not set by the user, the GAMS optCA value is used.

(default = GAMS optCA)

mip_integral_gap_rel (*double*)

The relative integrality gap stop tolerance. If not set by the user, the GAMS optCR value is used.

(default = GAMS optCR)

mip_lpalg (*integer*)

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`.

0 Knitro will 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

(default = 0)

mip_maxnodes (*integer*)

Specifies the maximum number of nodes explored (0 means no limit).

(default = 100000)

mip_maxsolves (*integer*)

Specifies the maximum number of subproblem solves allowed (0 means no limit).

(default = 200000)

mip_maxtime_cpu (*double*)

Specifies the cumulative CPU time limit, in seconds.

(default = 1e8)

mip_maxtime_real (*double)*

Specifies the cumulative real or wall-clock time limit, in seconds.

(default = 1e8)

mip_method (*integer)*

Specifies which method to use.

0 automatic

1 use the standard B&B method

2 use the hybrid Quesada-Grossman method (for convex, nonlinear problems only)

(default = 0)

mip_outinterval (*integer)*

Specifies node printing interval for `mip_outlevel` when `mip_outlevel`>`0`.

1 print output every node

2 print output every 2nd node

n print output every n'th node

(default = 10)

mip_outlevel (*integer)*

Specifies how much MIP information to print.

0 do not print any MIP node information

1 print one line of output for every node

(default = 1)

mip_rootalg (*integer)*

Specifies which algorithm to use for the root node solve.

0 Knitro will automatically try to choose the best algorithm based on the problem characteristics

1 Knitro will use the Interior/Direct algorithm

2 Knitro will use the Interior/CG algorithm

3 Knitro will use the Active Set algorithm

(default = 0)

mip_rounding (*integer)*

Specifies the rounding rule to apply.

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

(default = 0)

mip_selectrule (*integer)*

Specifies the select rule for choosing the next node in the tree.

0 automatic

1 search the tree using a depth first procedure

2 select the node with the best relaxation bound

3 use depth first unless pruned, then best bound

(default = 0)

mip_strong_candlim (*integer*)

Specifies the maximum number of candidates to explore for strong branching.

(default = 10)

mip_strong_level (*integer*)

Specifies the maximum number of tree levels on which to perform strong branching.

(default = 10)

mip_strong_maxit (*integer*)

Specifies the maximum number of iterations to allow for strong branching.

(default = 1000)

mip_terminate (*integer*)

Specifies conditions for terminating the MIP algorithm.

0 terminate at optimum

1 terminate at first integer feasible point

(default = 0)

ms_enable (*boolean*)

Toggles multi-start method.

(default = false)

ms_maxbndrange (*double*)

Maximum range to vary unbounded x when generating start points.

(default = 1e3)

ms_maxsolves (*integer*)

Specifies the maximum number of start points to try during multi-start.

0 Knitro sets the number based on problem size

n try exactly $n > 0$ start points

(default = 200000)

ms_maxtime_cpu (*double*)

Specifies the cumulative CPU time limit, in seconds.

(default = 1e8)

ms_maxtime_real (*double*)

Specifies the cumulative real or wall-clock time limit, in seconds.

(default = 1e8)

ms_startptrange (*double*)

Maximum range to vary all x when generating start points.

(default = 1e20)

ms_terminate (*integer*)

Specifies conditions for terminating the multi-start algorithm.

0 terminate after msmaxsolves

1 terminate at first local optimum (if before msmaxsolves)

2 terminate at first feasible solution (if before msmaxsolves)

(default = 0)

objrange (*double*)

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)

opttol (*double*)

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 (*double*)

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.

0 printing of all output is suppressed

1 print only summary information

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`

(default = 2)

pivot (*double*)

Specifies the initial pivot threshold used in the factorization routine. The value should be in the range $[0 \ldots 0.5]$ 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)

scale (*integer*)

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.

0 No scaling is performed.

1 The objective function and constraints may be scaled.

(default = 1)

soc (*integer*)

Specifies whether or not to try second order corrections (SOC). A second order correction may be beneficial for proglems with highly nonlinear constraints.

0 No second order correction steps are attempted

1 Second order correction steps may be attempted on some iterations.

2 Second order correction steps are always attempted if the original step is rejected and there are nonlinear constraints.

(default = 1)

xtol (*double*)

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 (1.1) are:

$$\begin{aligned}
\nabla_x \mathcal{L}(x,\lambda) = \nabla f(x) + \sum_{i=1\dots m} \lambda_i^c \nabla c_i(x) + \sum_{j=1\dots n} \lambda_j^b &= 0 && (6.3)\\
\lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))] &= 0, \quad i = 1\dots m && (6.4)\\
\lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)] &= 0, \quad j = 1\dots n && (6.5)\\
c_i^L \le c_i(x) &\le c_i^U, \quad i = 1\dots m && (6.6)\\
b_j^L \le x_j &\le b_j^U, \quad j = 1\dots n && (6.7)\\
\lambda_i^c &\ge 0, \quad i \in \mathcal{I},\ c_i^L = -\infty,\ c_i^U \text{finite} && (6.8)\\
\lambda_i^c &\le 0, \quad i \in \mathcal{I},\ c_i^U = \infty,\ c_i^L \text{finite} && (6.9)\\
\lambda_j^b &\ge 0, \quad j \in \mathcal{B},\ b_j^L = -\infty,\ b_j^U \text{finite} && (6.10)\\
\lambda_j^b &\le 0, \quad j \in \mathcal{B},\ b_j^U = \infty,\ b_j^L \text{finite} && (6.11)
\end{aligned}$$

where $\mathcal{I}$ and $\mathcal{B}$ represent the sets of indices corresponding to the general inequality constraints and non-fixed variable bound constraints respectively, λ_i^c is the Lagrange multiplier corresponding to constraint $c_i(x)$, and λ_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, as indicated in (6.8) - (6.11).

In KNITRO we define the feasibility error (`FeasErr`) at a point x^k to be the maximum violation of the constraints (6.7), (6.8), i.e.,

$$\text{FeasErr} = \max_{i=1\dots m,\ j=1\dots n} (0, (c_i^L - c_i(x^k)), (c_i(x^k) - c_i^U), (b_j^L - x_j^k), (x_j^k - b_j^U)), \tag{6.12}$$

while the optimality error (`OptErr`) is defined as the maximum violation of the first three conditions (6.3) - (6.5):

$$\text{OptErr} = \max_{i=1\dots m,\ j=1\dots n} (\|\nabla_x \mathcal{L}(x^k, \lambda^k)\|_\infty, \lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))], \lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)]). \tag{6.13}$$

The remaining conditions on the sign of the multipliers (6.8) - (6.11) 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

$$\begin{aligned} \tau_1 &= \max(1, (c_i^L - c_i(x^0)), (c_i(x^0) - c_i^U), (b_j^L - x_j^0), , (x_j^0 - b_j^U))), && (6.14) \\ \tau_2 &= \max(1, \|\nabla f(x^k)\|_\infty), && (6.15) \end{aligned}$$

where x^0 represents the initial point.

For unconstrained problems, the scaling (6.15) is not effective since $\|\nabla f(x^k)\|_\infty \to 0$ as a solution is approached. Therefore, for unconstrained problems only, the following scaling is used in the termination test

$$\tau_2 = \max(1, \min(|f(x^k)|, \|\nabla f(x^0)\|_\infty)), \tag{6.16}$$

in place of (6.15).

KNITRO stops and declares `Locally optimal solution found` if the following stopping conditions are satisfied:

$$\begin{aligned} \text{FeasErr} &\le \max(\tau_1 * \texttt{feastol}, \texttt{feastolabs}) && (6.17) \\ \text{OptErr} &\le \max(\tau_2 * \texttt{opttol}, \texttt{opttolabs}) && (6.18) \end{aligned}$$

where `feastol`, `opttol`, `feastolabs` and `opttolabs` are user-defined options (see section 2).

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 `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 $\tau_2 \to \infty$ while the optimality error (6.13) stays bounded, condition (6.18) 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 (1.1) 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 (1.1) (upper bound if maximizing). If a feasible point is found for problem (1.1) that satisfies the discrete restrictions on the variables, then this provides an upper bound on the optimal objective value of problem (1.1) (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 (1.1) 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

$$f(x_I) - f(x_R) \le \texttt{mip\_integral\_gap\_abs} \tag{6.19}$$

or

$$f(x_I) - f(x_R) \le \texttt{mip\_integral\_gap\_rel} \cdot \max(1, |f(x_I)|), \tag{6.20}$$

where `mip_integral_gap_abs` and `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 `mip_integral_gap_abs` and `mip_integral_gap_rel` are taken from the GAMS options optCA and optCR, but an explicit setting of `mip_integral_gap_abs` or `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 2) 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.

`Iinf` 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 6 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. In this case Knitro provides an option which does not require that the Hessian itself be computed and stored, but rather only Hessian times vector products are stored. 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

$$cl + tol \leq c(x) \leq cu - tol \tag{9.21}$$

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 (9.21) 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 [5].

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 (1.1*c*) 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 (1.1a) 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

$$f(x) = \tfrac{1}{2} \textstyle\sum_{j=1}^{q} r_j(x)^2.$$

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

$$J(x) = \left[\frac{\partial r_j}{\partial x_i} \right]_{\substack{j = 1,2,\ldots,q \\ i = 1,2,\ldots,n}} .$$

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 `algorithm=1`, and will be very similar to the classical Levenberg-Marquardt method when `algorithm=2`. For a discussion of these methods see, for example, [7].

KNITRO **References**

[1] R. H. Byrd, J. Ch. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000.

[2] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. On the convergence of successive linear-quadratic programming algorithms. Technical Report OTC 2002/5, Optimization Technology Center, Northwestern University, Evanston, IL, USA, 2002.

[3] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming, Series B*, 100(1):27–48, 2004.

[4] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[5] R. H. Byrd, J. Nocedal, and R. A. Waltz. Feasible interior methods using slacks for nonlinear optimization. *Computational Optimization and Applications*, 26(1):35–61, 2003.

[6] Harwell Subroutine Library. *A catalogue of subroutines (HSL 2002).* AEA Technology, Harwell, Oxfordshire, England, 2002.

[7] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Series in Operations Research. Springer, 1999.

[8] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. Technical Report 2003-6, Optimization Technology Center, Northwestern University, Evanston, IL, USA, June 2003. To appear in Mathematical Programming, Series A.

LGO

János D. Pintér, Pintér Consulting Services, Inc. Halifax, NS, Canada, B3M 1J2
jdpinter@hfx.eastlink.ca, http://www.dal.ca/~jdpinter

Contents

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 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:

- Bound-constrained local search, based on the use of an exact penalty function (EPM)
- Constrained local search, based on a generalized reduced gradient approach (GRG).

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.

[1] Also see `http://www.dal.ca/~jdpinter/l_s_d.html`

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. Tractable model sizes depend only on the available hardware, although LGO has a 2000 variable, 2000 constraint size limit.

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:

```
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

```
optname = value
```

with one option on each line. For example, one can write

```
opmode = 1
```

This specifies LGO to use global branch and bound search and the built-in local search methods.

The GAMS/LGO options are divided into two main categories:

- General options
- Limits and tolerances

2.1 General Options

Option	Description	Default
`opmode`	Specifies the search mode used. `0`: Local search from the given nominal solution without a preceding local search (LS) `1`: Global branch-and-bound search and local search (BB+LS) `2`: Global adaptive random search and local search (GARS+LS) `3`: Global multistart random search and local search (MS+LS)	`3`

Option	Description	Default
`tlimit`	Time limit in seconds. This is equivalent to the GAMS option `reslim`. If specified, this overrides the GAMS `reslim` option.	`1000`
`log_time`	Iteration log time interval in seconds. Log output occurs every `log_time` seconds.	`0.5`
`log_iter`	Iteration log time interval. Log output occurs every `log_iter` iterations.	`10`
`log_err`	Iteration log error output. Error reported (if applicable) every `log_err` iterations.	`10`
`debug`	Debug option. Prints out complete LGO status report to listing file. `0`: No `1`: Yes	`0`
`callConopt`	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.	`5`
`help`	Prints out all available GAMS/LGO solver options in the log and listing files.	

Note that the local search operational mode (`opmode 0`) is the fastest, and that it will work for convex, as well as for some non-convex models. If the model has a highly non-convex (multiextremal) structure, then at least one of the global search modes should be used. It may be a good idea to apply all three global search modes, to verify the global solution, or perhaps to find alternative good solutions. Usually, `opmode 3` is the safest (and slowest), since it applies several local searches; opmodes 1 and 2 launch only a single local search from the best point found in the global search phase.

2.2 Limits and Tolerances

Option	Description	Default
`g_maxfct`	Maximum number of merit (model) 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.	`500(n+m)`
`max_nosuc`	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.	`100(n+m)`
`penmult`	Constraint penalty multiplier. Global merit function is defined as objective + the constraint violations weighted by `penmult`.	`100`
`acc_tr`	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`.	`-1.0E+10`
`fct_trg`	Target objective function value (partial stopping criterion in internal local search phase).	`-1.0E+10`
`fi_tol`	Local search (merit function improvement) tolerance.	`1.0E-06`
`con_tol`	Maximal constraint violation tolerance in local search.	`1.0E-06`
`kt_tol`	Kuhn-Tucker local optimality condition violation tolerance.	`1.0E-06`
`irngs`	Random number seed.	`0`
`var_lo`	Smallest (default) lower bound, unless set by user.	`-1.0E+06`
`var_up`	Largest (default) upper bound, unless set by user.	`1.0E+6`
`bad_obj`	Default value for objective function, if evaluation errors occur.	`1.0E+8`

Note that if model-specific information is known (more sensible target objective/merit function value, tolerances, tighter variable bounds), then such information should always be used, since it may help to solve the model far more efficiently than the usage of 'blind' defaults.

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.

Wright, M H, Numerical Methods for Nonlinearly Constrained Optimization.
PhD thesis, Stanford University, 1976.
$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/LGODOC/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:

Field	Description
`Iter`	Current iteration.
`Objective`	Current objective function value.
`SumInf`	Sum of constraint infeasibilities.
`MaxInf`	Maximum constraint infeasibility.
`Seconds`	Current elapsed time in seconds.
`Errors`	Number of errors and type. Type can either be `D/E:` Evaluation error `B:` Bound violation.

```
      Iter      Objective     SumInf   MaxInf     Seconds  Errors
       500   4.515428E-01   5.76E-02  5.8E-02       0.007
      1000   6.700705E-01   5.03E-05  5.0E-05       0.014
      1500   2.765930E+00   6.25E-04  6.2E-04       0.020
      2000   2.710653E+00   1.55E-02  1.6E-02       0.026
      2500   4.016702E+00   1.44E-02  1.4E-02       0.032
      3000   4.865399E+00   2.88E-04  2.9E-04       0.038
      3500   4.858826E+00   3.31E-03  3.3E-03       0.044
      4000   1.106472E+01   1.53E-02  1.5E-02       0.050
      4500   1.595505E+01   1.56E-06  1.6E-06       0.055
      5000   1.618715E+01   2.17E-05  2.2E-05       0.062
      5500   1.618987E+01   3.45E-04  3.5E-04       0.067
      6000   1.985940E+01   4.03E-04  4.0E-04       0.074
      6500   1.624319E+01   5.64E-03  5.6E-03       0.079
      7000   1.727653E+01   8.98E-05  9.0E-05       0.086
      7500   1.727033E+01   3.03E-03  3.0E-03       0.091
      7840   2.933167E-02   0.00E+00  0.0E+00       0.097
```

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.

Illustrative References

R. Horst and P. M. Pardalos, Editors (1995) *Handbook of Global Optimization.* Vol. 1. Kluwer Academic Publishers, Dordrecht.

P. M. Pardalos and H. E. Romeijn, Editors (2002) *Handbook of Global Optimization.* Vol. 2. Kluwer Academic Publishers, Dordrecht.

J. D. Pintér (1996) *Global Optimization in Action*, Kluwer Academic Publishers, Dordrecht.

J. D. Pintér (2001) *Computational Global Optimization in Nonlinear Systems: An Interactive Tutorial*, Lionheart Publishing, Atlanta, GA.

J. D. Pintér (2002) Global optimization: software, tests and applications. Chapter 15 (pp. 515-569) in: Pardalos and Romeijn, Editors, *Handbook of Global Optimization.* Vol. 2. Kluwer Academic Publishers, Dordrecht.

LINDOGlobal

Lindo Systems, Inc.

Contents

1 Introduction

GAMS/LINDOGlobal finds guaranteed globally optimal solutions to general nonlinear problems with continuous and/or discrete variables. GAMS/LINDOGlobal 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/LINDOGlobal 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/LINDOGlobal like a regular nonlinear solver.

GAMS/LINDOGlobal 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.

LINDOGlobal automatically detects problem type and uses an appropriate solver, e.g., if you submit an LP model to LINDOGlobal, 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, LINDOGlobal 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, LINDOGlobal 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. The GAMS/LINDOGlobal license does not include the Barrier solver option. LINDOGlobal 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/LINDOGlobal

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

```
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 LINGOGlobal. You use NLPEC to translate complementarities into standard mathematical statements, e.g. h*y = 0, and then use LINDOGlobal as the DNLP(Discontinuous Nonlinear) solver to solve the translated model. The following little GAMS model illustrates:

```
   $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=lindoglobal;
   solve example using mpec min f;
```

2 Supported nonlinear functions

GAMS/LINDOGlobal 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/LINDOGlobal output

The log output below is obtained for the NLP model mhw4d.gms from the GAMS model library using LINDOs global solver.

```
LINDOGLOBAL            8Apr10 23.4.0 WIN 17007.17023 VS8 x86/MS Windows

   LINDOGLOBAL Driver
   Lindo Systems Inc, www.lindo.com

Lindo API version 6.0.1.406 built on Mar 17 2010 22:51:09
Barrier Solver Version 5.0.0.127, Nonlinear Solver Version 3.14T
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
 EP_MULTIPLY  EP_POWER  EP_SQR

Starting GOP presolve ...

Pre-check unboundedness
Computing reduced bound...
Searching for an initial solution...

Initial objective value: 0.029311

Starting reformulation ...

Model                    Input     Operation    Atomic      Convex
Number of variables  :       5             6        20          20
Number of constraints:       3             4        18          46
integer variables    :       0             0         0           0
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
```

```
  #ITERs    TIME(s)    LOWER BOUND     UPPER BOUND    BOXES

       1         0  -3.167052e+022   +2.931083e-002        1
      41         0  +2.630106e-002   +2.931083e-002       35 (*I)


Terminating global search ...

 Global optimum found
 Objective value             :        0.0293108307216
 Best Bound                  :        0.0283736126217
 Factors (ok,stb)            :                    892 (100.00,99.78)
 Simplex iterations          :                   4060
 Barrier iterations          :                      0
 Nonlinear iterations        :                    627
 Box iterations              :                     41
 Total number of boxes       :                     35
 Max. Depth                  :                      8
 Total time (sec.)           :                      0
```

After determining the different kinds of nonlinear operators LINDOGlobal 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.

4 Summary of LINDOGlobal Options

LINDOGlobal 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 LINDOGlobal parameters, categorized by type, along with their brief descriptions. A more detailed description is given in the section that follows.

4.1 LINDOGlobal Options File

In order to set LINDOGlobal options, you need to set up an option file *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 LINDOGlobal 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 *lindoglobal.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
MAXLOCALSEARCH      3

* Set an overall time limit of 200 secs.
SOLVER_TIMLMT       200
```

4.2 General Options

DECOMPOSITION_TYPE	decomposition to be performed on a linear or mixed integer model
SOLVER_IUSOL	flag for computing basic solution for infeasible model
SOLVER_TIMLMT	time limit in seconds for continous solver
SOLVER_FEASTOL	feasibility tolerance
SOLVER_RESTART	starting basis flag
SOLVER_OPTTOL	dual feasibility tolerance

4.3 LP Options

SPLEX_SCALE	scaling flag
SPLEX_ITRLMT	simplex iteration limit
SPLEX_PPRICING	pricing option for primal simplex method
SPLEX_REFACFRQ	number of simplex iterations between two consecutive basis re-factorizations
PROB_TO_SOLVE	controls whether the explict primal or dual form of the given LP problem will be solved
SPLEX_DPRICING	pricing option for dual simplex method
SPLEX_DUAL_PHASE	controls the dual simplex strategy
LP_PRELEVEL	controls the amount and type of LP pre-solving
SOLVER_CUTOFFVAL	solver will exit if optimal solution is worse than this
SOLVER_USECUTOFFVAL	flag for using cutoff value

4.4 MIP Options

MIP_TIMLIM	time limit in seconds for integer solver
MIP_AOPTTIMLIM	time in seconds beyond which the relative optimality tolerance will be applied
MIP_LSOLTIMLIM	time limit until finding a new integer solution
MIP_PRELEVEL	controls the amount and type of MIP pre-solving at root node
MIP_NODESELRULE	specifies the node selection rule
MIP_INTTOL	absolute integer feasibility tolerance
MIP_RELINTTOL	relative integer feasibility tolerance
MIP_RELOPTTOL	MIP relative optimality tolerance
MIP_PEROPTTOL	MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds
MIP_MAXCUTPASS_TOP	number passes to generate cuts on the root node

MIP_MAXCUTPASS_TREE	number passes to generate cuts on the child nodes
MIP_ADDCUTPER	percentage of constraint cuts that can be added
MIP_ADDCUTPER_TREE	percentage of constraint cuts that can be added at child nodes
MIP_MAXNONIMP_CUTPASS	number of passes allowed in cut-generation that does not improve current relaxation
MIP_CUTLEVEL_TOP	combination of cut types to try at the root node when solving a MIP
MIP_CUTLEVEL_TREE	combination of cut types to try at child nodes in the B&B tree when solving a MIP
MIP_CUTTIMLIM	time to be spent in cut generation
MIP_CUTDEPTH	threshold value for the depth of nodes in the B&B tree
MIP_CUTFREQ	frequency of invoking cut generation at child nodes
MIP_HEULEVEL	specifies heuristic used to find integer solution
MIP_CUTOFFOBJ	defines limit for branch & bound
MIP_USECUTOFFOBJ	flag for using branch and bound limit
MIP_STRONGBRANCHLEVEL	depth from the root in which strong branching is used
MIP_TREEREORDERLEVEL	tree reordering level
MIP_BRANCHDIR	first branching direction
MIP_TOPOPT	optimization method to use when there is no previous basis
MIP_REOPT	optimization method to use when doing reoptimization
MIP_SOLVERTYPE	optimization method to use when solving mixed-integer models
MIP_KEEPINMEM	flag for keepin LP bases in memory
MIP_BRANCHRULE	rule for choosing the variable to branch
MIP_REDCOSTFIX_CUTOFF	cutoff value as a percentage of the reduced costs
MIP_ADDCUTOBJTOL	required objective improvement to continue generating cuts
MIP_HEUMINTIMLIM	minimum time in seconds to be spent in finding heuristic solutions
MIP_BRANCH_PRIO	controls how variable selection priorities are set and used
MIP_SCALING_BOUND	maximum difference between bounds of an integer variable for enabling scaling
MIP_PSEUDOCOST_WEIGT	weight in pseudocost computations for variable selection
MIP_LBIGM	Big-M value used in linearizing nonlinear expressions
MIP_DELTA	near-zero value used in linearizing nonlinear expressions
MIP_DUAL_SOLUTION	flag for computing dual solution of LP relaxation
MIP_BRANCH_LIMIT	limit on the total number of branches to be created during branch and bound
MIP_ITRLIM	iteration limit for branch and bound
MIP_AGGCUTLIM_TOP	max number of constraints involved in derivation of aggregation cut at root node
MIP_AGGCUTLIM_TREE	max number of constraints involved in derivation of aggregation cut at tree nodes
MIP_ANODES_SWITCH_DF	threshold on active nodes for switching to depth-first search
MIP_ABSOPTTOL	MIP absolute optimality tolerance
MIP_MINABSOBJSTEP	value to update cutoff value each time a mixed integer solution is found
MIP_PSEUDOCOST_RULE	specifies the rule in pseudocost computations for variable selection
MIP_USE_ENUM_HEU	frequency of enumeration heuristic
MIP_PRELEVEL_TREE	amount and type of MIP pre-solving at tree nodes
MIP_REDCOSTFIX_CUTOFF_TREE	cutoff value as a percentage of the reduced costs at tree nodes
MIP_USE_INT_ZERO_TOL	controls if all MIP calculations would be based on absolute integer feasibility tolarance
MIP_USE_CUTS_HEU	controls if cut generation is enabled during MIP heuristics
MIP_BIGM_FOR_INTTOL	threshold for which coefficient of a binary variable would be considered as big-M
MIP_STRONGBRANCHDONUM	minimum number of variables to try the strong branching on
MIP_MAKECUT_INACTIVE_COUNT	threshold for times a cut could remain active after successive reoptimization
MIP_PRE_ELIM_FILL	controls fill-in introduced by eliminations during pre-solve

4.5 NLP Options

NLP_SOLVE_AS_LP	flag indicating if the nonlinear model will be solved as an LP
NLP_SOLVER	type of nonlinear solver
NLP_SUBSOLVER	type of nonlinear subsolver
NLP_PSTEP_FINITEDIFF	value of the step length in computing the derivatives using finite differences
NLP_DERIV_DIFFTYPE	flag indicating the technique used in computing derivatives with finite differences
NLP_FEASTOL	feasibility tolerance for nonlinear constraints
NLP_REDGTOL	tolerance for the gradients of nonlinear functions
NLP_USE_CRASH	flag for using simple crash routines for initial solution
NLP_USE_STEEPEDGE	flag for using steepest edge directions for updating solution
NLP_USE_SLP	flag for using sequential linear programming step directions for updating solution
NLP_USE_SELCONEVAL	flag for using selective constraint evaluations for solving NLP
NLP_PRELEVEL	controls the amount and type of NLP pre-solving
NLP_ITRLMT	nonlinear iteration limit
NLP_LINEARZ	extent to which the solver will attempt to linearize nonlinear models
NLP_STARTPOINT	flag for using initial starting solution for NLP
NLP_QUADCHK	flag for checking if NLP is quadratic
NLP_AUTODERIV	defining type of computing derivatives
NLP_MAXLOCALSEARCH	maximum number of local searches
NLP_USE_LINDO_CRASH	flag for using advanced crash routines for initial solution
NLP_STALL_ITRLMT	iteration limit before a sequence of non-improving NLP iterations is declared as stalling
NLP_AUTOHESS	flag for using Second Order Automatic Differentiation for solving NLP

4.6 Global Options

OPTTOL	optimality tolerance
FLTTOL	floating-point tolerance
BOXTOL	minimal width of variable intervals
WIDTOL	maximal width of variable intervals
DELTATOL	delta tolerance in GOP convexification
BNDLIM	max magnitude of variable bounds used in GOP convexification
TIMLIM	time limit in seconds for GOP branch-and-bound
OPTCHKMD	criterion used to certify the global optimality
BRANCHMD	direction to branch first when branching on a variable
MAXWIDMD	maximum width flag for the global solution
PRELEVEL	amount and type of GOP presolving
POSTLEVEL	amount and type of GOP postsolving
BBSRCHMD	node selection rule in GOP branch-and-bound
DECOMPPTMD	decomposition point selection rule in GOP branch-and-bound
ALGREFORMMD	algebraic reformulation rule for a GOP
RELBRNDMD	reliable rounding in the GOP branch-and-bound
USEBNDLIM	max magnitude of variable bounds flag for GOP convexification
BRANCH_LIMIT	limit on the total number of branches to be created in GOP tree
CORELEVEL	strategy of GOP branch-and-bound
OPT_MODE	mode for GOP optimization
HEU_MODE	heuristic used in global solver
SUBOUT_MODE	substituting out fixed variables

LSOLBRANLIM	branch limit until finding a new nonlinear solution
USEGOP	use global optimization

4.7 Link Options

CHECKRANGE	calculate feasible range for variables
WRITEMPI	write MPI file of processed model

5 Detailed Descriptions of LINDOGlobal Options

DECOMPOSITION_TYPE ***(integer)***

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

SPLEX_SCALE ***(integer)***

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

SPLEX_ITRLMT ***(integer)***

This is a limit on the number of iterations the solver will perform before terminating. If this value is a nonegative 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 = GAMS IterLim)

SPLEX_PPRICING ***(integer)***

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)***

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)

PROB_TO_SOLVE (*integer*)

This flag controls whether the explict 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

SPLEX_DPRICING (*integer*)

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*)

This controls the dual simplex strategy, single-phase versus two-phase.

(default = 0)

0	Solver decides
1	Single-phase
2	Two-phase

LP_PRELEVEL (*integer*)

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

SOLVER_IUSOL (*integer*)

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 appopriate 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_TIMLMT (*integer*)

This is a time limit in seconds for the LP solver. The default value of -1 imposes no time limit.

(default = GAMS ResLim)

SOLVER_CUTOFFVAL (*real*)

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*)

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_RESTART (*integer*)

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_OPTTOL (*real*)

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_USECUTOFFVAL (*integer*)

This is a flag for the parameter SOLVER_CUTOFFVAL

(default = 0)

0	Do not use cutoff value
1	Use cutoff value

NLP_SOLVE_AS_LP (*integer*)

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_SOLVER (*integer*)

This value determines the type of nonlinear solver.

(default = 7)

4	Solver decides
7	Uses CONOPTs reduced gradient solver
9	Uses CONOPT with multistart feature enabled

NLP_SUBSOLVER (*integer*)

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_PSTEP_FINITEDIFF (*real*)

This controls the value of the step length in computing the derivatives using finite differences.

(default = 5e-7)

NLP_DERIV_DIFFTYPE (*integer*)

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_FEASTOL (*real*)

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_REDGTOL (*real*)

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_USE_CRASH (*integer*)

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_STEEPEDGE (*integer*)

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

NLP_USE_SLP (*integer*)

This is a flag indicating if sequential linear programming step directions should be used in updating the solution.

(default = 1)

0 Do not use sequential linear programming step directions
1 Use sequential linear programming step directions

NLP_USE_SELCONEVAL (*integer*)

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_PRELEVEL (*integer*)

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_ITRLMT (*integer*)

This controls the iteration limit on the number of nonlinear iterations performed.

(default = GAMS IterLim)

NLP_LINEARZ (*integer*)

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_STARTPOINT (*integer*)

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_QUADCHK (*integer*)

This is a flag indicating if the nonlinear model should be examined to check if it is a quadratic model.

(default = 0)

0	Do not check if NLP is quadratic
1	Check if NLP is quadratic

NLP_AUTODERIV (*integer*)

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_MAXLOCALSEARCH *(integer)*

This controls the maximum number of local searches (multistarts) when solving a NLP using the multistart solver.

(default = 5)

NLP_USE_LINDO_CRASH *(integer)*

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_STALL_ITRLMT *(integer)*

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_AUTOHESS *(integer)*

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

MIP_TIMLIM *(integer)*

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_TIMLMT will be applied to each continuous subproblem solve. If the value of this parameter is greater than 0, then the value of SOLVER_TIMLMT 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_AOPTTIMLIM *(integer)*

This is the time in seconds beyond which the relative optimality tolerance, MIP_PEROPTTOL will be applied.

(default = 100)

MIP_LSOLTIMLIM *(integer)*

(default = -1)

MIP_PRELEVEL *(integer)*

This controls the amount and type of MIP pre-solving at root node.

(default = 510)

+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 Maximum pass

MIP_NODESELRULE (*integer*)

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_INTTOL (*real*)

An integer variable is considered integer feasible if the absolute difference from the nearest integer is smaller than this.

(default = 1e-6)

MIP_RELINTTOL (*real*)

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*)

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 = 1e-5)

MIP_PEROPTTOL (*real*)

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_MAXCUTPASS_TOP (*integer*)

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*)

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_ADDCUTPER (*real*)

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*)

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_MAXNONIMP_CUTPASS (*integer*)

This controls the maximum number of passes allowed in cut-generation that does not improve the current relaxation.

(default = 3)

MIP_CUTLEVEL_TOP (*integer*)

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 = 6142)

+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

MIP_CUTLEVEL_TREE (*integer*)

This controls the combination of cut types to try at child nodes in the B&B tree when solving a MIP.

(default = 4094)

+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

MIP_CUTTIMLIM ***(integer)***

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_CUTDEPTH ***(integer)***

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)***

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_HEULEVEL ***(integer)***

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_CUTOFFOBJ ***(real)***

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_USECUTOFFOBJ ***(integer)***

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_STRONGBRANCHLEVEL ***(integer)***

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_TREEREORDERLEVEL (*integer*)

This specifies the tree reordering level.

(default = 10)

MIP_BRANCHDIR (*integer*)

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_TOPOPT (*integer*)

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_REOPT (*integer*)

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_SOLVERTYPE (*integer*)

This specifies the optimization method to use when solving mixed-integer models.

(default = 0)

0 Solver decides

1 Use B&B only

2 Use Enumeration and Knapsack solver only

MIP_KEEPINMEM (*integer*)

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_BRANCHRULE (*integer*)

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

MIP_REDCOSTFIX_CUTOFF (*real*)

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_ADDCUTOBJTOL (*real*)

This specifies the minimum required improvement in the objective function for the cut generation phase to continue generating cuts.

(default = 1.5625e-5)

MIP_HEUMINTIMLIM (*integer*)

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_BRANCH_PRIO (*integer*)

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_SCALING_BOUND (*integer*)

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-andbound tree.

(default = 10000)

MIP_PSEUDOCOST_WEIGT (*real*)

This specifies the weight in pseudocost computations for variable selection.

(default = 1.5625e-05)

MIP_LBIGM (*real*)

This refers to the Big-M value used in linearizing nonlinear expressions.

(default = 10000)

MIP_DELTA (*real*)

This refers to a near-zero value used in linearizing nonlinear expressions.

(default = 1e-6)

MIP_DUAL_SOLUTION (*integer*)

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_BRANCH_LIMIT (*integer)*

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_ITRLIM (*integer)*

This is the iteration limit for branch-and- bound. The default value is .1, which means no iteration limit is imposed. If the iteration limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

(default = -1)

MIP_AGGCUTLIM_TOP (*integer)*

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)*

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)*

This specifies the threshold on active nodes for switching to depth-first search rule.

(default = 50000)

MIP_ABSOPTTOL (*real)*

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 = 0)

MIP_MINABSOBJSTEP (*real)*

This specifies the value to update the cutoff value each time a mixed integer solution is found.

(default = 0)

MIP_PSEUDOCOST_RULE (*integer)*

This specifies the rule in pseudocost computations for variable selection.

(default = 0)

MIP_USE_ENUM_HEU (*integer)*

This specifies the frequency of enumeration heuristic.

(default = 4)

MIP_PRELEVEL_TREE (*integer)*

This controls the amount and type of MIP pre-solving at tree nodes.

(default = 174)

+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	Maximum pass

MIP_REDCOSTFIX_CUTOFF_TREE (*real*)

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_USE_INT_ZERO_TOL (*integer*)

This flag controls if all MIP calculations would be based on the integrality tolarance specified by MIP_INTTOL.

(default = 0)

0	Do not base MIP calculations on MIP_INTTOL
1	Base MIP calculations on MIP_INTTOL

MIP_USE_CUTS_HEU (*integer*)

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_BIGM_FOR_INTTOL (*real*)

This value specifies the threshold for which the coefficient of a binary variable would be considered as big-M (when applicable).

(default = 1e8)

MIP_STRONGBRANCHDONUM (*integer*)

This value specifies the minimum number of variables, among all the candidates, to try the strong branching on.

(default = 3)

MIP_MAKECUT_INACTIVE_COUNT (*integer*)

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_PRE_ELIM_FILL (*integer*)

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)

OPTTOL (*real*)

This value is the GOP optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

(default = 1e-6)

FLTTOL *(real)*

This value is the GOP floating-point tolerance. It specifies the maximum rounding errors in the floating-point computation.

(default = 1e-10)

BOXTOL *(real)*

This value specifies the minimal width of variable intervals in a box allowed to branch.

(default = 1e-6)

WIDTOL *(real)*

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 MAXWIDMD is set at 1.

(default = 1e-4)

DELTATOL *(real)*

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)

BNDLIM *(real)*

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)

TIMLIM *(integer)*

This is the time limit in seconds for GOP branch-and-bound.

(default = GAMS ResLim)

OPTCHKMD *(integer)*

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 OPTTOL at the global optimum. When its value is 1, the relative deviation of objective lower and upper bounds should be smaller than OPTTOL at the global optimum.

(default = 1)

BRANCHMD *(integer)*

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
5	Relative violation between the function and its convex envelope at the convex minimum

MAXWIDMD *(integer)*

This is the maximum width flag for the global solution. The GOP branch-andbound may continue contracting a box with an incumbent solution until its maximum width is smaller than WIDTOL.

(default = 0)

0	The maximum width criterion is suppressed
1	The maximum width criterion is performed

PRELEVEL *(integer)*

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

POSTLEVEL *(integer)*

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

BBSRCHMD *(integer)*

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

DECOMPPTMD *(integer)*

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

ALGREFORMMD *(integer)*

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

RELBRNDMD *(integer)*

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

USEBNDLIM (*integer*)

This value is a flag for the parameter 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

BRANCH_LIMIT (*integer*)

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)

CORELEVEL (*integer*)

This controls the strategy of GOP branch-and-bound procedure.

(default = 14)

+2 LP convex relaxation

+4 NLP solving

+8 Box Branching

OPT_MODE (*integer*)

This specifies the mode for GOP optimization.

(default = 1)

HEU_MODE (*integer*)

This specifies the heuristic used in the global solver to find 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

SUBOUT_MODE (*integer*)

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

LSOLBRANLIM (*integer*)

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)

CHECKRANGE *(string)*

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)

USEGOP *(integer)*

This value determines whether the global optimization will be used.

(default = 1)

0 Do not use global optimization

1 Use global optimization

WRITEMPI *(string)*

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.

LogMIP

Contents

1 Introduction

LogMIP 1.0 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 1.0 has been developed by A. Vecchietti, J.J. Gil and L. Catania at INGAR (Santa Fe-Argentina) and Ignacio E. Grossmann at Carnegie Mellon University (Pittsburgh-USA) and is composed of:

- a language compiler for the declaration and definition of disjunctions and logic constraints
- solvers for linear and non-linear disjunctive models (lmbigm, lmchull)

Those components are linked to GAMS. Both parts are supersets of GAMS language and solvers respectively. LogMIP is not independent of GAMS. Besides the disjunction and logic constraints declaration and definition, LogMIP needs the declaration and definitions of scalars, sets, tables, variables, constraints, equations, etc. made in GAMS language for the specifications and solution of a disjunctive problem.

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/`
- Documentation: `http://www.logmip.ceride.gov.ar/sites/default/files/logmip_manual.pdf`

MILES

Thomas F. Rutherford, University of Colorado

Contents

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.

1 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:

$$\begin{array}{rl}
\text{Given:} & F : R^n \to R^n \ , \quad \ell, u \in R^n \\[1em]
\text{Find:} & z, w, v \in R^n \\[1em]
\text{such that} & F(z) = w - v \\
& \ell \le z \le u, \quad w \ge 0, \quad v \ge 0 \\
& w^T(z - \ell) = 0 \ , \quad v^T(u - z) = 0.
\end{array}$$

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:

I 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).

II 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 "`OPTION MCP = 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 2 introduces the Newton algorithm. Section 3 describes the implementation of Lemke's algorithm which is used to solve linear subproblems. Section 4 defines switches and tolerances which may be specified using the options file. Section 5 interprets the run-time log file which is normally directed to the screen. Section 6 interprets the status file and the detailed iteration reports which may be generated. Section 7 lists and suggests remedies for abnormal termination conditions.

2 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 z in the neighborhood of $\hat{z}$, 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: $z^{k+1} = z^k + \lambda d^k$. An Armijo or "back-tracking" line search initially considers $\lambda = 1$. If $\|F(z^z + \lambda d^k)\| \leq \|F(z^k)\|$, the step size λ is adopted, otherwise is multiplied by a positive factor α, $\alpha < 1$, and the convergence test is reapplied. This procedure is repeated until either an improvement results or $\lambda < \underline{\lambda}$. When $\underline{\lambda} = 0$, a positive step is taken provided that [2]:

$$\frac{d}{d\lambda}\|F(z^k + \lambda d^k)\| < 0.$$

[1]There is one other MCP solver available through GAMS: PATH (Ferris and Dirkse, 1992)

[2]α and $\underline{\lambda}$ correspond to user-specified tolerances `DMPFAC` and `MINSTP`, respectively

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:

I Does the linearized problem always have a solution?

II If the linearized problem has a solution, does Lemke's algorithm find it?

III 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"[3].

The answer to question (III) 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 $z, \epsilon(z)$, is based on a measure of the extent to which z, w and v violate applicable upper and lower bounds and complementarity conditions.

Evaluating Convergence

Let δ_i^L and δ_i^U be indicator variables for whether z_i is off its lower or upper bound. These are defined as [4]:

$$\delta_i^L = min(1, (z_i - \ell_i)^+) \quad \text{and} \quad \delta_i^U = min(1, (u_i - z_i)^+).$$

[3]Kaneko (1978) provides some convergence theory for the linearized subproblem

[4]In the following $x^+ = max(x, 0)$

Given z, MILES uses the value of $F(z)$ to implicitly define the slack variables w and v:

$$w_i = F_i(z)^+ \ , \quad v_i = \Big(-F_i(z)\Big)^+ .$$

There are two components to the error term associated with index i, one corresponding to z_i's violation of upper and lower bounds:

$$\varepsilon_i^B = (z_i - u_i)^+ + (\ell_i - z_i)^+$$

and another corresponding to violations of complementarity conditions:

$$\varepsilon_i^C = \delta_i^L w_i + \delta_i^U v_i .$$

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 $+\infty$.[5]

3 Lemke's Method with Implicit Bounds

A mixed linear complementarity problem has the form:

$$\begin{array}{rl} \text{Given:} & M \in R^{n\times n}, \qquad q, \ell, u \in R^n \\ \\ \text{Find:} & z, w, v \in R^n \\ \\ \text{such that} & Mz + q = w - v, \\ & \ell \le z \le u, \quad w \ge 0, \quad v \ge 0, \\ & w^T(z-\ell) = 0 \ , \quad v^T(u-z) = 0. \end{array}$$

In the Newton subproblem at iteration k, the LCP data are given by $q = F(z^k) - \nabla F(z^k) z^k$ and $M = \nabla F(z^k)$.

The Working Tableau

In MILES, the pivoting scheme for solving the linear problem works with a re-labeled linear system of the form:

$$Bx^B + Nx^N = q,$$

where $x^B \in R^n, \quad x^N \in R^{2n}$, and the tableau $[B|N]$ is a conformal "complementary permutation" of $[-M|\ I\ |-I]$. That is, every column i in B must either be the ith 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.

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:

$$\underline{x}_i^B = \begin{cases} \ell_i, & \texttt{if} \quad x_i^B = z_i \\ 0, & \texttt{if} \quad x_i^B = w_i \ \texttt{or} \ v_i, \end{cases}$$

$$\overline{x}_i^B = \begin{cases} u_i, & \texttt{if} \quad x_i^B = z_i \\ \infty, & \texttt{if} \quad x_i^B = w_i \ \texttt{or} \ v_i \end{cases}$$

[5] Parameter p may be selected with input parameter `NORM`. The default value for p is $+\infty$.

The values of the non-basic variables x_i^N and x_{i+n}^N are determined by the assignment of x_i^B:

$$x_i^B = \begin{cases} z_i & \Rightarrow \begin{cases} x_i^N & = & w_i & = & 0 \\ x_{i+n}^N & = & v_i & = & 0 \end{cases} \\ \\ w_i & \Rightarrow \begin{cases} x_i^N & = & z_i & = & \ell_i \\ x_{i+n}^N & = & v_i & = & 0 \end{cases} \\ \\ v_i & \Rightarrow \begin{cases} x_i^N & = & w_i & = & 0 \\ x_{i+n}^N & = & z_i & = & u_i. \end{cases} \end{cases}$$

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:

$$x^B = B^{-1}\Big(q - Nx^N\Big).$$

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.

Initialization

Let B^0 denote the initial basis matrix [6] . The initial values for basic variables are then:

$$\hat{x}^B = (B^0)^{-1}(q - N\ \hat{x}^N).$$

If $\underline{x}^B \leq \hat{x}^B \leq \bar{x}^B$, then the initial basis is feasible and the complementarity problem is solved [7]. Otherwise, MILES introduces an artificial variable z_0 and an artificial column h. Basic variables are then expressed as follows:

$$x^B = \hat{x}^B - \tilde{h}z_0,$$

where $\tilde{h}$ is the "transformed artificial column" (the untransformed column is $h = B^0\tilde{h}$). The coefficients of $\tilde{h}$ are selected so that:

I The values of "feasible" basis variables are unaffected by z_0: ($\underline{x}_i^B \leq x_i^B \leq \bar{x}_i^B \Longrightarrow \tilde{h}_i = 0$).

II The "most infeasible" basic variable ($i = p$) is driven to its upper or lower bound when $z_0 = 1$:

$$\tilde{h}_p = \begin{cases} \hat{x}_p^B - \bar{x}_p^B, & if \quad \tilde{x}_p^B > \bar{x}_p^B \\ \\ \hat{x}_p^B - \underline{x}_p^B, & if \quad \tilde{x}_p^B < \underline{x}_p^B \ . \end{cases}$$

III All other *infeasible* basic variables assume values between their upper and lower bounds when z_0 increases to 1:

$$x_i^B = \begin{cases} 1 + \underline{x}_i^B, & if \quad \underline{x}_i^B > -\infty, \quad \bar{x}_i^B = +\infty \\ \\ \frac{\bar{x}_i^B + \underline{x}_i^B}{2}, & if \quad \underline{x}_i^B > -\infty, \quad \bar{x}_i^B < +\infty \\ \\ \bar{x}_i^B - 1, & if \quad \underline{x}_i^B = -\infty, \quad \bar{x}_i^B < +\infty \ . \end{cases}$$

[6] In Miles, B^0 is chosen using the initially assigned values for z. When $z_i \leq \ell_i$, then $x_i^B = w_i$; when $z_i \geq u_i$, then $x_i^B = v_i$; otherwise $x_i^B = z_i$.

[7] 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.

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 ℓ_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

N	Exiting Variable	Entering Variable	Change in Non-basic Values
I	z_i at lower bound	w_i increases from 0	$x_i^N = z_i = \ell_i$
II	z_i at upper bound	v_i increases from 0	$x_{i+n}^N = z_i = u_i$
III	w_i at 0	z_i increases from ℓ_i	$x_i^N = x_{i+n}^N = 0$
IV	v_i at 0	z_i decreases from u_i	$x_i^N = x_{i+n}^N = 0$

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_i'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 z_i is increased from ℓ_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 [8].

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.

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.)

When MILES encounters a secondary ray, a restart procedure is invoked in which the set of basic variables associated with z_0^* are reinstalled. 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.

[8] 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.

4 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* [9]. The following is a typical options file:

```
BEGIN SPECS
               ITLIMT = 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:

Termination control

Option	Description	Default
`CONTOL`	is the convergence tolerance. Whenever an iterate is encountered for which $\epsilon(z)$ <`CONTOL`, the algorithm terminates. This corresponds to the GAMS/MINOS parameter "Row tolerance".	`1.0E-6`
`ITLIMT`	is an upper bound on the number of Newton iterations. This corresponds to the GAMS/MINOS parameter "Major iterations".	`25`
`ITERLIM`	is an upper bound on the cumulative number of Lemke iterations. When MILES is invoked from within a GAMS program (either with MPSGE or GAMS/MCP), `<model>.ITERLIM` can be used to set this value. This corresponds to the GAMS/MINOS parameter "Iterations limit".	`1000`
`NORM`	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 $+\infty$.	`3`
`NRFMAX`	establishes an upper bound on the number of factorizations in any LCP. This avoids wasting lots of CPU if a subproblem proves difficult to solve.	`3`
`NRSMAX`	sets an upper bound on the number of restarts which the linear subproblem solver will undertake before giving up.	`1`

Basis factorization control

Option	Description	Default
`FACTIM`	indicates the maximum number of CPU seconds between recalculation of the basis factors.	`120.0`
`INVFRQ`	determines the maximum number of Lemke iterations between recalculation of the basis factors. This corresponds to the GAMS/MINOS parameter "Factorization frequency".	`200`
`ITCH`	indicates the frequency with which the factorization is checked. The number refers to the number of basis replacements operations between refinements. This corresponds to the GAMS/MINOS parameter "Check frequency".	`30`

[9]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.

Pivot Selection

Option	Description	Default
PLINFY	is the value assigned for "plus infinity" ("+INF" in GAMS notation).	1.D20
ZTOLPV	is the absolute pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.	3.644E-11 (EPS**(2./3.)) [10]
ZTOLRP	is the relative pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.	3.644E-11 (EPS**(2./3.))
ZTOLZE	is used in the subproblem solution to determine when any variable has exceeded an upper or lower bound. This corresponds to GAMS/MINOS parameter "Feasibility tolerance".	1.E-6

Linearization and Data Control

Option	Description	Default
SCALE	invokes row and column scaling of the LCP tableau in every iteration. This corresponds, roughly, to the GAMS/MINOS switch "scale all variables".	.TRUE.
ZTOLDA	sets a lower bound on the magnitude of nonzeros recognized in the linearization. All coefficients with absolute value less than ZTOLDA are ignored.	1.483E-08 (EPS**(1/2))

Newton Iteration Control

Option	Description	Default
DMPFAC	is the Newton damping factor, represented by α above.	0.5
MINSTP	is the minimum Newton step length, represented by $\underline{\lambda}$ above.	0.03

Output Control

Option	Description	Default
INVLOG	is a switch which requests LUSOL to generate a report with basis statistics following each refactorization.	.TRUE.
LCPDMP	is a switch to generate a printout of the LCP data after scaling.	.FALSE.
LCPECH	is a switch to generate a printout of the LCP data before scaling, as evaluated.	.FALSE.
LEVOUT	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 "Print level"	0
PIVLOG	is a switch to generate a status file listing of the Lemke pivot sequence.	.FALSE.

LUSOL parameters

(as with MINOS 5.4, except LUSIZE)

Option	Description	Default
`LUSIZE`	is used to estimate the number of LU nonzeros which will be stored, as a multiple of the number of nonzeros in the Jacobian matrix.	`10`
`LPRINT`	is the print level, < 0 suppresses output. `= 0` gives error messages. `= 1` gives debug output from some of the other routines in LUSOL. ≥ 2 gives the pivot row and column and the no. of rows and columns involved at each elimination step in lu1fac.	`0`
`MAXCOL`	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.	`5`
`ELMAX1`	is the maximum multiplier allowed in L during factor.	`10.0`
`ELMAX2`	is the maximum multiplier allowed in L during updates.	`10.0`
`SMALL`	is the absolute tolerance for treating reals as zero. IBM double: 3.0d-13	`EPS**(4./5.)`
`UTOL1`	is the absolute tol for flagging small diagonals of U. IBM double: 3.7d-11	`EPS**(2./3.)`
`UTOL2`	is the relative tol for flagging small diagonals of U. IBM double: 3.7d-11	`EPS**(2./3.)`
`USPACE`	is a factor which limits 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.	`3.0`
`DENS1`	is the density at which the Markowitz strategy should search maxcol columns and no rows.	`0.3`
`DENS2`	is 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.	`0.6`

5 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 ($\epsilon_i^B + \epsilon_i^C$). 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, amd final deviation. The final message reports the solution status. In this case, the model has been successfully processed ("Solved.").

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

Solved.

Work space allocated           --    0.01 Mb

Initial deviation ........ 5.750E+02    W_02
Convergence tolerance .... 1.000E-06

   0    5.75E+02    1.00E+00 (W_02    )
   1    2.51E+01    1.00E+00 (P_01    )
   2    4.53E+00    1.00E+00 (P_01    )
   3    1.16E+00    1.00E+00 (P_01    )
   4    3.05E-01    1.00E+00 (P_01    )
   5    8.08E-02    1.00E+00 (P_01    )
   6    2.14E-02    1.00E+00 (P_01    )
   7    5.68E-03    1.00E+00 (P_01    )
   8    1.51E-03    1.00E+00 (P_01    )
   9    4.00E-04    1.00E+00 (P_01    )
  10    1.06E-04    1.00E+00 (P_01    )
  11    2.82E-05    1.00E+00 (P_01    )
  12    7.47E-06    1.00E+00 (P_01    )
  13    1.98E-06    1.00E+00 (P_01    )
  14    5.26E-07    1.00E+00 (P_01    )

Major iterations ........    14
Lemke pivots ............    23
Refactorizations ........    15
Deviation .............. 5.262E-07

Solved.
```

6 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. Tables 3-6 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 $z-$ <name i >, $w-$ <name i >, and $v-$ <name i >, as shown in the $COLUMNS section. The nonzeros for $w-$ <> and $v-$ <> variables are not shown because they are assumed to be $-/+I$.

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 $\tilde{h}$ 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 $h = B^0\tilde{h}$ 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 $\tilde{h}$ and h.).

Table 6 displays the final section of the status file. At the top of page 6 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)

```
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

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).   5.000E-01
    Minimum step length ....(MINSTP).   1.000E-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
===============================================================
ITER   DEVIATION        STEP
   0    3.25E+02    1.00E+00 (P_01    )
===============================================================
```

Table 4 Status File with Debugging Output (page 2 of 4)

```
 Iteration   0 values.
               ROW           Z                         F
             --------    ------------             ------------
             X_01_01  L   0.00000E+00             -7.75000E-01
             X_01_02  L   0.00000E+00             -8.47000E-01
             X_01_03  L   0.00000E+00             -8.38000E-01
             X_02_01  L   0.00000E+00             -7.75000E-01
             X_02_02  L   0.00000E+00             -8.38000E-01
             X_02_03  L   0.00000E+00             -8.74000E-01
             W_01     L   0.00000E+00              3.25000E+02
             W_02     L   0.00000E+00              5.75000E+02
             P_01         1.00000E+00             -3.25000E+02
             P_02         1.00000E+00             -3.00000E+02
             P_03         1.00000E+00             -2.75000E+02
 ==================================
 Function Evaluation, Iteration:  0
 ==================================
$ROWS:
X_01_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_02    -1.53000004E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_03    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_02    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_03    -1.25999998E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
W_01       -3.25000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
W_02       -5.75000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
P_01        3.25000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_02        3.00000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_03        2.75000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
     ...    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
$COLUMNS:
Z-X_01_01   W_01     -1.00000000E+00
            P_01      1.00000000E+00
Z-X_01_02   W_01     -1.00000000E+00
            P_02      1.00000000E+00
Z-X_01_03   W_01     -1.00000000E+00
            P_03      1.00000000E+00
Z-X_02_01   W_02     -1.00000000E+00
            P_01      1.00000000E+00
Z-X_02_02   W_02     -1.00000000E+00
            P_02      1.00000000E+00
Z-X_02_03   W_02     -1.00000000E+00
            P_03      1.00000000E+00
Z-W_01      X_01_01   1.00000000E+00
            X_01_02   1.00000000E+00
            X_01_03   1.00000000E+00
Z-W_02      X_02_01   1.00000000E+00
            X_02_02   1.00000000E+00
            X_02_03   1.00000000E+00
Z-P_01      X_01_01  -1.00000000E+00
            X_02_01  -1.00000000E+00
Z-P_02      X_01_02  -1.00000000E+00
            X_02_02  -1.00000000E+00
Z-P_03      X_01_03  -1.00000000E+00
            X_02_03  -1.00000000E+00
       ...       ...  0.00000000E+00
```

Table 5 Status File with Debugging Output (page 3 of 4)

```
SCALING LCP DATA
-------
            MIN ELEM    MAX ELEM       MAX COL RATIO
AFTER  0    1.00E+00    1.00E+00                1.00
AFTER  1    1.00E+00    1.00E+00                1.00
AFTER  2    1.00E+00    1.00E+00                1.00
AFTER  3    1.00E+00    1.00E+00                1.00
SCALING RESULTS:

      A(I,J) <= A(I,J) * 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
```

Table 6 Status File with Debugging Output (page 4 of 4)

```
                        LEMKE PIVOT STEPS
                        =================
ITER  STATUS   Z%   Z0     ERROR   INFEAS.  ---- PIVOTS ----  IN       OUT
   1    1.00    0  1.000                    3.E+02 1.E+00  1  Z0       W    9
   2    1.00    9  1.000                    1.E+00 1.E+00  2  Z    9   W    1
   3    1.00   18  0.997                    9.E-01 9.E-01  1  Z    1   W   10
   4    1.00   27  0.997                    1.E+00 1.E+00  1  Z   10   W    2
   5    1.00   36  0.996                    9.E-01 4.E-01  1  Z    2   W   11
   6    1.00   45  0.996                    1.E+00 1.E+00  1  Z   11   W    6
   7    1.00   55  0.479                    2.E+00 1.E+00  1  Z    6   W    7
   8    1.00   64  0.479                    1.E+00 1.E+00  1  Z    7   W    4
   9    1.00   73  0.000                    1.E+00 1.E+00  2  Z    4   W    8
  10    1.00   73  0.000                    1.E-03 2.E-03  1  V    8   Z0

================================
Convergence Report, Iteration  1
===============================================================
ITER   DEVIATION        STEP
   0    3.25E+02    1.00E+00
   1    1.14E-13    1.00E+00 (W_02    )
===============================================================

Iteration   1 values.

               ROW           Z                        F
             --------    ------------             ------------
             X_01_01      2.50000E+01             -8.32667E-17
             X_01_02      3.00000E+02             -5.55112E-17
             X_01_03  L   0.00000E+00              3.60000E-02
             X_02_01      3.00000E+02             -8.32667E-17
             X_02_02  L   0.00000E+00              8.99999E-03
             X_02_03      2.75000E+02              2.77556E-17
             W_01         1.00000E+00             -1.13687E-13
             W_02         1.00000E+00              1.13687E-13
             P_01         1.22500E+00              0.00000E+00
             P_02         1.15300E+00              0.00000E+00
             P_03         1.12600E+00              0.00000E+00

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.
```

7 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 [11].

`Failure to converge.` Two successive iterates are identical - the Newton search direction is not defined. This should normally not occur.

`Inconsistent parameters ZTOLZO, ZTOLZE.` `ZTOLZO` determines the smallest value loaded into the cover vector h, whereas `ZTOLZE` is the feasibility tolerance employed in the Harris pivot selection procedure. If `ZTOLZO < -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.

References

K.J. Anstreicher, J. Lee and T.F. Rutherford "Crashing a Maximum Weight Complementary Basis", Mathematical Programming. (1992)

A. Brooke, D. Kendrick, and A. Meeraus "GAMS: A User's Guide", Scientific Press, (1987).

R.W. Cottle and J.S. Pang "The Linear Complementarity Problem", Academic Press, (1992).

J.E. Dennis and R.B. Schnabel "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice-Hall (1983).

S. Dirkse "Robust solution of mixed complementarity problems", Computer Sciences Department, University of Wisconsin (1992).

B.C. Eaves, "A locally quadratically convergent algorithm for computing stationary points," Tech. Rep., Department of Operations Research, Stanford University, Stanford, CA (1978).

P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright "Maintaining LU factors of a general sparse matrix", Linear Algebra and its Applications 88/89, 239-270 (1991).

C.B. Garcia and W.I. Zangwill "Pathways to Solutions, Fixed Points, and Equilibria", Prentice-Hall (1981)

P. Harker and J.S. Pang "Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications", Mathematical Programming 48, pp. 161-220 (1990).

W.W. Hogan, "Energy policy models for project independence," Computation and Operations Research 2 (1975) 251-271.

[11] 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.

N.H. Josephy, ”Newton’s method for generalized equations”, Technical Summary Report #1965, Mathematical Research Center, University of Wisconsin - Madison (1979).

I. Kaneko, ”A linear complementarity problem with an n by 2n ’P’- matrix”, Mathematical Programming Study 7, pp. 120-141, (1978).

C.E. Lemke ”Bimatrix equilibrium points and mathematical programming”, Management Science 11, pp. 681-689, (1965).

L. Mathiesen, ”Computation of economic equilibria by a sequence of linear complementarity problems”, Mathematical Programming Study 23 (1985).

P.V. Preckel, ”NCPLU Version 2.0 User’s Guide”, Working Paper, Department of Agricultural Economics, Purdue University, (1987).

W.H. Press, B.P.Flannery, S.A. Teukolsky, W.T. Vetterling ”Numerical Recipes: The Art of Scientific Computing”, Cambridge University Press (1986).

J.M. Ortega and W.C. Rheinboldt, ”Iterative Solution of Nonlinear Equations in Several Variables”, Academic Press (1970).

S.M. Robinson, ”A quadratically-convergent algorithm for general nonlinear programming problems”, Mathematical Programming Study 3 (1975).

T.F. Rutherford ”Extensions of GAMS for variational and complementarity problems with applications in economic equilibrium analysis”, Working Paper 92-7, Department of Economics, University of Colorado (1992a).

T.F. Rutherford ”Applied general equilibrium modeling using MPS/GE as a GAMS subsystem”, Working Paper 92-15, Department of Economics, University of Colorado (1992b).

Table 7 Transport Model in GAMS/MCP (page 1 of 2)

```
*==>TRNSP.GMS
option mcp=miles;

$TITLE  LP TRANSPORTATION PROBLEM FORMULATED AS A ECONOMIC EQUILIBRIUM
*       ==============================================================
*       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:
*              Dantzig, G B., Linear Programming and Extensions
*              Princeton University Press, Princeton, New Jersey, 1963,
*              Chapter 3-3.
*
*              This formulation is described in detail in Chapter 2
*              (by Richard E. Rosenthal) of GAMS: A Users' Guide.
*              (A Brooke, D Kendrick and A Meeraus, The Scientific Press,
*              Redwood City, California, 1988.
*
   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
                     NEW-YORK       CHICAGO      TOPEKA
       SEATTLE          2.5           1.7          1.8
       SAN-DIEGO        2.5           1.8          1.4  ;

  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 * 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;
```

MINOS

Bruce A. Murtagh; Graduate School of Management, Macquarie University, Sydney, Australia
Michael A. Saunders, Walter Murray; Department of EESOR, Stanford University, CA
Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA

Contents

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.

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:

```
  option nlp=minos;   { or lp or dnlp or rminlp }
```

or

```
  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:

```
  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:

```
  > 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:

NLP

$$\begin{array}{lll} \underset{x,y}{\text{minimize}} & F(x) + c^T x + d^T y & (1) \\ \text{subject to} & f(x) + A_1 y \sim b_1 & (2) \\ & A_2 x + A_3 y \sim b_2 & (3) \\ & \ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u & (4) \end{array}$$

where the vectors c, d, b_1, b_2, ℓ, 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 ℓ and u may be $-\infty$ or $+\infty$ respectively, in accordance with the GAMS use of `-INF` and `+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

$$\boxed{\text{LP} \qquad \begin{array}{ll} \underset{x}{\text{minimize}} & c^T x \\ \text{subject to} & Ax + Is = 0 \\ & \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u \end{array}}$$

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 $\begin{pmatrix} A & I \end{pmatrix}$. 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: $\ell_j - \delta < x_j < \ell_j$ or $u_j < x_j < u_j + \delta$ where δ is a *feasibility tolerance* (typically 10^{-6}). In rare cases, even nonbasic variables might lie outside their bounds by as much as δ.

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, 11, 12]). 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[14] combined with a *quasi-Newton* algorithm that is described in [8]. 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 δ, 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_s 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^Tg 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

$$R^TRq = -Z^Tg$$

where g is a gradient of $F(x)$, Z^Tg 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 $R^TR \approx Z^THZ$ 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 *line search* is then performed to find an approximate solution to the one-dimensional problem

$$\begin{aligned} &\underset{\alpha}{\text{minimize}} \;\; F(x+\alpha p) \\ &\text{subject to } \; 0 < \alpha < \beta \end{aligned}$$

where β 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 α (see [6]). The number of nonlinear function evaluations required may be influenced by setting the **Linesearch tolerance**, as discussed in Section 9.

As a linear programming solver, an equation $B^T\pi = gB$ is solved to obtain the *dual variables* or *shadow prices* π 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 $Z^T g = gs - s^T\pi$; 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 π.)

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 [13], see [9]. 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 λ_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

$$\boxed{\begin{array}{lll} \underset{x,y}{\text{minimize}} & F(x) + c^Tx + d^Ty - \lambda_k^T(f - f') + 0.5\rho(f-f')^T(f-f') & (5) \\ \text{subject to} & f' + A_1 y \sim b_1 & (6) \\ & A_2 x + A_3 y \sim b_2 & (7) \\ & \ell \le \begin{pmatrix} x \\ y \end{pmatrix} \le u & (8) \end{array}}$$

The objective function (5) is called an *augmented Lagrangian*. The scalar ρ is a *penalty parameter*, and the term involving ρ 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):

Example	
	$\underset{r,a,b}{\text{minimize}}\quad r$
	subject to $\;(x_i - a)^2 + (y_i - b)^2 \le r^2,\;\; r \ge 0.$

This problem can be modeled in GAMS as follows.

```
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

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

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.

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 $z = x^{f(y)}$. In that case it is advised to introduce an extra variable and equation:

$$\begin{gathered} z = x^{\vartheta} \\ \vartheta = f(y) \\ \vartheta \geq \varepsilon \end{gathered}$$

(Note that the function `SQR(x)` does not require x to be positive).

If the model produces *function evaluation errors* adding bounds is prefered 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

```
   variables x,y,z;
   equations e1,e2;
   e1..z =e= y;
   e2..y =e= sqr(1+x);
   model m /all/;
   option nlp=minos;
   solve m using nlp minimizing z;
```

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 §4.1).

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 §1.2.

NLP

This option selects the NLP solver. Example: `option NLP=MINOS;`. See also §1.2.

DNLP

Selects the DNLP solver for models with discontinuous or non-differentiable functions. Example: `option DNLP=MINOS;`. See also §1.2.

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 §1.2.

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 §1.2.

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 $x < \varepsilon$ (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 $\sqrt{x}, x < 0$ 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 §2.1 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:

```
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 §4.1.

m.reslim

Sets the resource or time limit. Overrides the global resource limit. Example: `m.reslim=600;` The default is 1000 seconds. See also §4.1.

m.bratio

Sets the basis acceptance test parameter. Overrides the global setting. Example: `m.bratio=1.0;` The default is 0.25. See also §4.1.

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 §9. 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. (The defaults are given in the Section 7.) 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

Debug level	Controls amounts of output information
Log Frequency	Frequency of iteration log information
Print level	Amount of output information
Scale, print	Causes printing of the row and column-scales
Solution No/Yes	Controls printing of final solution
Summary frequency	Controls information in summary file

6.2 Options Affecting Tolerances

Crash tolerance	crash tolerance
Feasibility tolerance	Variable feasibility tolerance for linear constraints
Line search tolerance	Accuracy of step length location during line search
LU factor tolerance	Tolerances during LU factorization
LU update tolerance	
LU Singularity tolerance	
Optimality tolerance	Optimality tolerance
Pivot Tolerance	Prevents singularity
Row Tolerance	Accuracy of nonlinear constraint satisfaction at optimum
Subspace tolerance	Controls the extent to which optimization is confined to the current set of basic and superbasic variables

6.3 Options Affecting Iteration Limits

Iterations limit	Maximum number of minor iterations allowed
Major iterations	Maximum number of major iterations allowed
Minor iterations	Maximum number of minor iterations allowed between successive linearizations of the nonlinear constraints

6.4 Other Algorithmic Options

Check frequency	frequency of linear constraint satisfaction test
Completion	accuracy level of sub-problem solution
Crash option	Perform crash
Damping parameter	See Major Damping Parameter
Expand frequency	Part of anti-cycling procedure
Factorization frequency	Maximum number of basis changes between factorizations
Hessian dimension	Dimension of reduced Hessian matrix
Lagrangian	Determines linearized sub-problem objective function
LU pivoting	Pivoting strategy in LUSOL
Major damping parameter	Forces stability between subproblem solutions
Minor damping parameter	Limits the change in x during a line search
Multiple price	Pricing strategy
Partial Price	Level of partial pricing
Penalty Parameter	Value of ρ in the modified augmented Lagrangian
Radius of convergence	Determines when ρ will be reduced
Scale option	Level of scaling done on the model
Start assigned nonlinears	Affects the starting strategy during cold start
Superbasics limit	Limits storage allocated for superbasic variables
Unbounded objective value	Detects unboundedness in nonlinear problems
Unbounded step size	Detects unboundedness in nonlinear problems
Verify level	Finite-difference check on the gradients
Weight on linear objective	Invokes the composite objective technique

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 $\lambda_k = 0$ and $\rho = 0$ 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 $\lambda_0 = 0$, 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 $\rho = 100.0/m_1$ where m_1 is the number of nonlinear constraints. For later subproblems, ρ is reduced in stages when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many times it is safe to specify $\lambda = 0$, 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 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 s 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 for each case, and the appearing messages are explained.

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...

    Itn  ninf      sinf       objective
    100     3 2.283E-01 -2.51821463E+04
    200     0 0.000E+00  2.02819284E+04
    300     0 0.000E+00  1.54107277E+04
    400     0 0.000E+00  1.40211808E+04
    500     0 0.000E+00  1.33804183E+04
    600     0 0.000E+00  1.27082709E+04

 EXIT - Optimal Solution found, objective:        12657.77

--- Restarting execution
--- turkpow.gms(205) 0 Mb
--- Reading solution for model turkey
--- turkpow.gms(230) 3 Mb
*** Status: Normal completion
```

The first line that is written by MINOS is the version string: `MINOS-Link May 25, 2002 WIN.M5.M5 20.6 023.046.040.VIS GAMS/MINOS 5.5`. 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-zeroes
---   706 nl-code  65 nl-non-zeroes
--- 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
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           --    0.82 Mb

  Reading Rows...
  Reading Columns...
  Reading Instructions...

    Itn  ninf      sinf       objective
    100     0 0.000E+00  1.71416714E+03
    200     0 0.000E+00  1.73483184E+03

 EXIT - Optimal Solution found, objective:        1735.570

--- Restarting execution
--- weapons.gms(68) 0 Mb
--- Reading solution for model war
--- weapons.gms(77) 3 Mb
*** Status: Normal completion
```

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

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           --    1.48 Mb

  Reading Rows...
  Reading Columns...
  Reading Instructions...

 Major minor  step     objective  Feasible Optimal  nsb   ncon penalty BSswp
     1     2T 0.0E+00  1.91724E+02 1.8E+02 2.0E-01    0      1 1.0E+00     0
     2    90  1.0E+00  1.91735E+02 1.5E-03 7.6E+00    0      3 1.0E+00     0
     3     0  1.0E+00  1.91735E+02 1.3E-09 5.5E-06    0      4 1.0E+00     0
     4     0  1.0E+00  1.91735E+02 1.1E-12 2.8E-13    0      5 1.0E-01     0

 EXIT - Optimal Solution found, objective:        191.7346

--- Restarting execution
--- camcge.gms(450) 0 Mb
--- Reading solution for model camcge
*** Status: Normal completion
```

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 $\begin{pmatrix} B & S \end{pmatrix}$ (i.e. basic vs. superbasic) changes.

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 δ denotes machine precision (typically 10^{-15} or 10^{-16}). Options not specified will take the default values shown.

Check frequency i

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 Full
Completion 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 λ_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)

Crash option i

If a restart is not being performed, an initial basis will be selected from certain columns of the constraint matrix $\begin{pmatrix} A & I \end{pmatrix}$. The value of 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)

crash option 0

The initial basis will contain only slack variables: $B = I$

crash option 1

All columns of A are considered (except those excluded by the *Start assigned nonlinears* option).

crash option 2
Only the columns of A corresponding to the linear variables y will be considered.

crash option 3
Variables that appear nonlinearly in the objective will be excluded from the initial basis.

crash option 4
Variables that appear nonlinearly in the constraints will be excluded from the initial basis.

Crash tolerance r
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 $a_{i,j}$ in the column are ignored if $|a_{i,j}| < a_{\max} \times r$. To be meaningful, r should be in the range $0 \le r < 1$).
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, 4, -1. To help *CRASH* choose all m columns for the initial basis, we could specify *Crash tolerance* r for some value of $r > 0.25$.
(Default = 0.1)

Damping parameter r
See *Major Damping Parameter*.
(Default = 2.0)

Debug level i
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.
(Default = 0)

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 i
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 δ. Over a period of i iterations, the tolerance actually used by GAMS/MINOS increases from 0.5 δ to δ (in steps $0.5\delta/i$).
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 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*).
(Default = 10000).

Factorization frequency i

At most i basis changes will occur between factori-zations 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.

(Default = 100 (50 for NLP's))

Feasibility tolerance r

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, $F(x)$ and $f(x)$ 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 $= 10^{-6}$)

Hessian dimension r

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 $Z^T HZ \approx R^T R$. 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,

$$R = \begin{pmatrix} R_r & 0 \\ & D \end{pmatrix}$$

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 $r^2/2$, which is substantial if r is as large as 200 (say). In general, r should be slight 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 = Superbasics limit)

Iterations limit i
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 Yes
Lagrangian No
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)

Linesearch tolerance r
For nonlinear problems, this controls the accuracy with which a step-length α is located in the one-dimensional problem

$$\begin{aligned} &\underset{\alpha}{\text{minimize}} \;\; F(x + \alpha p) \\ &\text{subject to} \;\; 0 < \alpha \le \beta \end{aligned}$$

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 in equation (5).)
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 i
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 = 1 or 100)

LU factor tolerance r_1
LU update tolerance r_2
The first two tolerances affect the stability and sparsity of the basis factorization $B = LU$ during re-factorization and updates respectively. The values specified must satisfy $r_i \ge 1.0$. The matrix L is a

product of matrices of the form:

$$\begin{pmatrix} I & \\ \mu & I \end{pmatrix}$$

where the multipliers μ will satisfy $|\mu| < r_i$.

I The default values $r_i = 10.0$ usually strike a good compromise between stability and sparsity.

II For large and relatively dense problems, $r_i = 25.0$ (say) may give a useful improvement in sparsity without impairing stability to a serious degree.

III For certain very regular structures (e.g., band matrices) it may be necessary to set r_1 and/or r_2 to values smaller than the default in order to achieve stability. For example, if the columns of A include a sub-matrix of the form:

$$\begin{pmatrix} 4 & -1 & \\ -1 & 4 & -1 \\ & -1 & 4 \end{pmatrix}$$

it would be judicious to set both r_1 and r_2 to values in the range $1.0 < r_i < 4.0$.

(Default values: $r_1 = 100.0$ (5 for NLP's), $r_2 = 10.0$ (5 for NLP's))

LU partial pivoting
LU rook pivoting
LU complete pivoting

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 $t_1 < 2.0$).
(Default = *LU partial pivoting*)

LU density tolerance r_1
LU singularity tolerance r_2

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 r_1 towards 1.0 may give slightly sparser factors, with a slight increase in factorization time. The singularity tolerance r_2 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{j,j}| \le r_2$ or $|U_{j,j}| < r_2 \max_i |U_{j,j}|$, 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 $r_2 = 1.0^{-5}$, say, may help cause a judicious change of basis.
(Default values: $r_1 = 0.5$, $r_2 = \varepsilon^{2/3} \approx 10^{-11}$)

Major damping parameter r

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, λ_k) and (x_{k+1}, λ_{k+1}). For example, the default value 2.0 prevents the relative change in either x_k or λ_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 λ_{k+1} are changed to $x_k + \sigma(x_{k+1} - x_k)$ and $\lambda_k + \sigma(\lambda_{k+1} - \lambda_k)$ for some step-length $\sigma < 1$. 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 ρ). (Skip this re-run in the case of nonlinear equations: there are no degrees of freedom and the value of ρ 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 σ 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.
(Default = 2.0)

Major iterations i

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 my not converge. The progress of the major iterations can be best monitored using *Print level* 0 (the default).
(Default = 50)

Minor damping parameter r

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

$$\underset{\alpha}{\text{minimize}} \; F(x + \alpha p)$$

is performed over the range $0 < \alpha \leq \beta$, where β is the step to the nearest upper or lower bound on x. Normally, the first step length tried is $a_1 = \min(1, \beta)$.
In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the components of r can lean 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 i

This is the maximum number of 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 i

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 be 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 r

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 j^{th} variable. a_j is the associated column of the constraint matrix (or Jacobian), and π 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 $d_j/||\pi|| \geq -r$ 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

$$\sigma = \sum_{i=1}^{m} |\pi_i|,\ ||\pi|| = \max\{\sigma/\sqrt{m}, 1\}$$

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 r.
(Default = 10^{-6})

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 basic or superbasic).
When $i = 1$, all columns of the constraints matrix $(A\ I)$ are searched.
Otherwise, A_j and I are partitioned to give i roughly equal segments A_j, I_j $(j = 1 \quad \text{to} \quad i)$. If the previous search was successful on A_{j-1}, I_{j-1}, the next search begins on the segments A_j, 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 for LPs, or 1 for NLPs)

Penalty Parameter r

This specifies the value of ρ in the modified augmented Lagrangian. It is used only when *Lagrangian = 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 ρ may be necessary of known to ensure convergence, *even for convex programs.* On the other hand, if ρ 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 *Iterations* and/or *major iterations* limit. If successive major iterations appear to be terminating with radically different solutions, the penalty parameter should be increased. (See also the *Major damping parameter.*) If there appears to be little progress between major iteration, it may help to reduce the penalty parameter.
(Default = $100.0/m_1$)

Pivot Tolerance r

Broadly speaking, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular. The default value of r should be satisfactory in most circumstances.

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.

For linear problems, elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.

For nonlinear problems, elements smaller than $r||p||$ are ignored.

It is common (on degenerate problems) for two or more variables to reach a bound at essentially the same time. In such cases, the *Feasibility tolerance* (say t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should not be specified.

To a lesser extent, the *Expand frequency* (say f) also provides some freedom to maximize pivot the element. Excessively large values of f should therefore not be specified .

(Default $= \varepsilon^{2/3} \approx 10^{-11}$)

Print level i

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

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 λ_k, the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if *Lagrangian=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   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.)

In MINOS 5.5 the log frequency relates to information written to the listing file only. This information is only visible when `OPTION SYSOUT=ON;` is used. Print level will no longer set the log frequency.

(Default = 0)

Radius of convergence r

This determines when the penalty parameter ρ will be reduced (if initialized to a positive value). Both the nonlinear constraint violation (see *ROWERR* below) and the relative change in consecutive Lagrange multiplier estimate must be less than r at the start of a major iteration before ρ 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 r

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 *ROWERR* be the maximum component of the residual vector $f(x) + A_1y - b_1$, normalized by the size of the solution. Thus

$$ROWERR = \frac{||f(x) + A_1y - b_1||_\infty}{1 + XNORM}$$

where *XNORM* is a measure of the size of the current solution (x, y). The solution is regarded acceptably feasible if $ROWERR \le r$.

If the problem functions involve data that is known to be of low accuracy, a larger *Row tolerance* may be appropriate.

(Default = 10^{-6})

Scale option i

Scaling done on the model.

(Default = 2 for LPs, 1 for NLPs)

Scale option 0
Scale no

No scaling. If storage is at a premium, this option should be used

Scale option 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.

Scale option 2
Scale nonlinear variables
Scale all 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 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.

Scale, print

This causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $a'_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables, and slacks are $l'_j = l_j/c(j), u'_j = u_j/c(j)$, 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 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:

$$\rho_j = \frac{\max_i |a_{ij}|}{\min_i |a_{ij}|}\ (a_{ij} \neq 0)$$

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.

If a *Scale option* has not already been specified, *Scale tolerance* sets the default scaling.

(Default = 0.9)

Solution yes
Solution no

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)

Start assigned nonlinears

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 non-default 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.
Note that the *first* and *fourth* keywords are significant.
(Default = superbasic)

Start assigned nonlinears superbasic

Specify *superbasic* for highly nonlinear models, as long as K is not too large (say $K < 100$) and the initial values are good.

Start assigned nonlinears basic

Specify *basic* for models that are essentially square (i.e., if there are about as many general constraints as variables).

Start assigned nonlinears nonbasic

Specify *nonbasic* if K is large.

Start assigned nonlinears 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 r

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).
r must be a real number in the range $0 < r \leq 1$.
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 i

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 $i = 10$ or 100 is often adequate to determine if the `SOLVE` is making progress. If *Print level* = 0, the default value of i is 100. If *Print level* > 0, the default value of i 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 = 1 or 100)

Superbasics limit i

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 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 $n_1 + 1$, 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.
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 = Hessian dimension)

Unbounded objective value r

These parameters are intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\underset{\alpha}{\text{minimize}}\ F(x + \alpha p)$$

If $|F|$ exceeds r or if α exceeds r_2, 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_1 can be made.
Unboundedness is x is best avoided by placing finite upper and lower bounds on the variables. See also the *Minor damping parameter.*
(Default = 10^{20})

Unbounded step size r

These parameters are intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\underset{\alpha}{\text{minimize}}\ F(x + \alpha p)$$

If α exceeds r, 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 x is best avoided by placing finite upper and lower bounds on the variables. See also the *Minor damping parameter.*
(Default = 10^{10})

Verify level i

This option refers to a finite-difference check on the gradients (first derivatives) computed by GAMS for each nonlinear function. GAMS computes gradients analytically, and the values obtained should normally be taken as correct.
Gradient verification occurs before the problem is scaled, and before the first basis is factorized. (Hence, it occurs before the basic variables are set to satisfy the general constraints $Ax + s = 0$.)
(Default = 0)

Verify level 0

Only a cheap test is performed, requiring three evaluations of the nonlinear objective (if any) and two evaluations of the nonlinear constraints. *Verify No* is an equivalent option.

Verify level 1

A more reliable check is made on each component of the objective gradient. *Verify objective gradients* is an equivalent option.

Verify level 2

A check is made on each column of the Jacobian matrix associated with the nonlinear constraints. *Verify constraint gradients* is an equivalent option.

Verify level 3
A detailed check is made on both the objective and the Jacobian. *Verify*, *Verify gradients*, and *Verify Yes* are equivalent options.

Verify level -1
No checking is performed.

Weight on linear objective r
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

$$\underset{x}{\text{minimize}}\ \sigma w(c^T x) + (\text{sum of infeasibilities})$$

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 GAMS `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^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 solution is to add bounds so that all functions can be properly evaluated. E.g. if you have an expression x^y, 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:

```
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;`.

MINOS References

[1] R. H. Bartels, *A stabilization of the simplex method*, Numerische Mathematik, 16 (1971), pp. 414–434.

[2] R. H. Bartels and G. H. Golub, *The simplex method of linear programming using the LU decomposition*, Communications of the ACM, 12 (1969), pp. 266–268.

[3] G. .B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

[4] W. C. Davidon, *Variable metric methods for minimization*, A.E.C. Res. and Develop. Report ANL-5990, Argonne National Laboratory, Argonne, IL., 1959.

[5] R. Fourer, *Solving staircase linear programs by the simplex method*, Mathematical Programming, 23 (1982), pp. 274–313.

[6] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, *Two step-length algorithms for numerical optimization*, Report SOL 79-25, Department of Operations Research, Stanford University, Stanford, California, 1979.

[7] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, *Maintaining factors of a general sparse matrix*, Linear Algebra and its Applications, 88/89 (1987), pp. 239–270.

[8] B. A. Murtagh and M. A. Saunders, *Large-scale linearly constrained optimization*, Mathematical Programming, 14 (1978), pp. 41–72.

[9] B. A. Murtagh and M. A. Saunders, *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Mathematic Programming Study, 16 (1982), Algorithms for Constrained Minimization of Smooth Nonlinear Function, pp. 84–117.

[10] B. A. Murtagh and M. A. Saunders, *MINOS 5.0 User‘s Guide*, Report SOL 83-20, Department of Operations Research, Stanford University, 1983 (Revised as MINOS 5.1 User‘s Guide, Report SOL 83-20R, 1987.)

[11] J. K. Reid, *Fortran subroutines for handling sparse linear programming bases*, Report R8269, Atomic Energy Research Establishment, Harwell, England, 1976.

[12] J. K. Reid, *A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases*, Mathematical Programming, 24 (1982), pp. 55–69.

[13] S. M. Robinson, *A quadratically convergent algorithm for general nonlinear programming problems*, Mathematical Programming 3 (1972), pp. 145–156.

[14] P. Wolfe, *The reduced-gradient method*, unpublished manuscript, RAND Corporation, 1962.

MOSEK

MOSEK ApS, C/O Symbion Science Park, Fruebjergvej 3, Box 16, 2100 Copenhagen Ø, Denmark

Contents

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 and convex quadratically constraint programs using an extremely efficient interior point algorithm. The interior point algorithm has many complex solver options which the user can specify to fine-tune the optimizer for a particular model.

Furthermore, MOSEK can solve generalized linear programs involving nonlinear conic constraints, convex quadratically constraint and general convex nonlinear programs.

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

1.1 Licensing

Licensing of GAMS/MOSEK is similar to other GAMS solvers. MOSEK is licensed in three different ways:

- **GAMS/MOSEK Base:**
  All continuous models
- **GAMS/MOSEK Extended:**
  Same as GAMS/MOSEK Base, but also the solution of models involving discrete variables.
- **GAMS/MOSEK Solver Link:**
  Users must have a seperate, licensed MOSEK system. For users who wish to use MOSEK within GAMS and also in other environments.

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/Undbounded 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

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b, \\ & x \geq 0 \end{array}$$

is the solution y satisfying:

$$A^T y \leq 0, \quad b^T y > 0$$

A dual infeasibility certificate is reported in the levels of the variables in the listing file. The dual infeasibility certificate x for the same minimization problem is

$$Ax = 0, \quad c^T x < 0$$

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 `(varible).LO` and `(variable).UP` suffixes and resolve.

For more information 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.

1.3.1 Parallelized Optimizers

Only the interior-point optimizer in MOSEK has 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 `MSK_IPAR_INTPNT_NUM_THREADS` sets the number of threads (and therefore the number of CPU's) that the interior point optimizer will use.

1.3.2 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 `MSK_IPAR_CONCURRENT_*` 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 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 MSK_IPAR_INFEAS_REPORT_LEVEL controls the amount info presented in the infeasibility report. We will use the trnsport.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.

Index    Name                   Lower bound      Upper bound      Dual lower       Dual upper
1        supply(seattle)        none             3.500000e+002    0.000000e+000    1.000000e+000
2        supply(san-diego)      none             6.000000e+002    0.000000e+000    1.000000e+000
3        demand(new-york)       5.200000e+002    none             1.000000e+000    0.000000e+000
4        demand(chicago)        4.800000e+002    none             1.000000e+000    0.000000e+000
5        demand(topeka)         4.400000e+002    none             1.000000e+000    0.000000e+000

The following bound constraints are involved in the infeasibility.

Index    Name                   Lower bound      Upper bound      Dual lower       Dual upper
```

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 seperable form, if it is possible. Consider the following example of minizing an objective function $f(x)$:

$$f(x) = \log(a' * x)$$

where $a \in \Re^n$ is a parameter and $x \in \Re^n$ 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:

$$\begin{array}{rcl} f(x) & = & \log(y) \\ y & = & a' * x \\ y & \geq & 0 \end{array}$$

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-seperable) form are more expensive. Finally, it reduces the amount of memory needed (see the section on "Memory Options")

2 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.

This section gives an overview of conic programming and how 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:

$$\begin{array}{lc} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq r^c, \\ & x \in [l^x, u^x] \\ & x \in C \end{array}$$

where $A \in \Re^{m \times n}$ is the constraint matrix, $x \in \Re^n$ the decision variable, and $c \in \Re^n$ the objective function cost coefficients. The vector $r^c \in \Re^m$ represents the right hand side and the vectors $l^x, u^x \in \Re^n$ are lower and upper bounds on the decision variable x.

[1]See M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, *Applications of second-order cone programming* Linear Algebra and its Applications, 284:193-228, Special Issue on Linear Algebra in Control, Signals and Image Processing. November, 1998.

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

$$S^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \; and \; S^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}. \tag{2.1}$$

Let x_{S^t} denote the variables x belonging to set S^t. Then define

$$C := \{x \in \Re^n : x_{S^t} \in C_t, t = 1, ..., k\} \tag{2.2}$$

where C_t must have one of the following forms:

- Quadratic cone: (also referred to as Lorentz or ice cream cone)

$$C_t = \left\{ x \in \Re^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{2.3}$$

- Rotated quadratic cone: (also referred to as hyberbolic constraints)

$$C_t = \left\{ x \in \Re^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \; x_1, x_2 \geq 0 \right\}. \tag{2.4}$$

These two types of cones allow the formulation of quadratic, quadratically constrained, and many other classes of nonlinear convex optimization problems.

2.2 Implemention of Conic Constraints in GAMS

GAMS handles conic equations using the `=C=` equation type. The conic cases are written as:

- Quadratic cone:

$$\texttt{x(`1`) = C = sum(i\$[not sameas(i,`1`)], x(i));} \tag{2.5}$$

- Rotated quadratic cone:

$$\texttt{x(`1`) + x(`2`) = C = sum(i\$[not sameas(i,`1`) and not sameas(i,`2`)], x(i));} \tag{2.6}$$

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:

$$\texttt{x(`1`) = G = sqrt[sum(i\$[not sameas(i,`1`)], sqr[x(i)])];} \tag{2.7}$$

- Rotated quadratic cone: $x('1')$ and $x('2')$ are positive variables

$$\texttt{2 * x(`1`) * x(`2`) = G = sum(i\$[not sameas(i,`1`) and not sameas(i,`2`)], sqr[x(i)]);} \tag{2.8}$$

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:

$$\begin{array}{lllll} \text{minimize} & \sum_i \frac{d_i}{x_i} & & & \\ \text{subject to} & a^t x & \leq & b & \\ & x_i & \in & [l_i, u_i], & l_i > 0,\ d_i \geq 0,\ i = 1, 2, ..., n \end{array} \tag{2.9}$$

where $x \in \Re^n$ is the decision variable, $d, a, l, u \in \Re^n$ parameters, and $b \in \Re$ a scalar parameter. The original model (2.9) can be written in GAMS using the equations:

```
defobj..      sum(n, d(n)/x(n)) =E= obj;
e1..          sum(n, a(n)*x(n)) =L= b;
Model orig /defobj, e1/;
x.lo(n) = l(n);
x.up(n) = u(n);
```

We can write an equivalent NLP formulation, replacing the objective function and adding another constraint:

$$\begin{array}{lllll} \text{minimize} & \sum_i d_i t_i & & & \\ \text{subject to} & a^t x & \leq & b & \\ & 2 t_i x_i & \geq & 2, & i = 1, ..., n \\ & x & \in & [l, u], & l > 0,\ d_i \geq 0 \end{array} \tag{2.10}$$

where $t \in \Re^n$ is a new decision variable. The GAMS formulation of this NLP (`model cnlp`) is:

```
defobjc..     sum(n, d(n)*t(n)) =E= obj;
e1..          sum(n, a(n)*x(n)) =L= b;
conenlp(n)..  2*t(n)*x(n) =G= 2;

Model cnlp /defobjc, e1, conenlp/;
x.lo(n) = l(n);
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 $z \in \Re^n, z_i = \sqrt{2}$, then we can reformulate the problem using conic constraints as:

$$\begin{array}{lllll} \text{minimize} & \sum_i d_i t_i & & & \\ \text{subject to} & a^t x & \leq & b & \\ & z_i & = & \sqrt{2} & \\ & 2 t_i x_i & \geq & z_i^2, & i = 1, ..., n \\ & x & \in & [l, u], & l > 0,\ d_i \geq 0 \end{array} \tag{2.11}$$

The GAMS formulation using conic equations `=C=` is:

```
defobjc..     sum(n, d(n)*t(n)) =E= obj;
e1..          sum(n, a(n)*x(n)) =L= b;
e2(n)..       z(n) =E= sqrt(2);
cone(n)..     x(n) + t(n) =C= z(n);

Model clp  /defobjc, e1, e2, cone/;
x.lo(n) = l(n);
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:

```
Set n / n1*n10 /;
Parameter d(n), a(n), l(n), u(n);
Scalar b;

d(n) = uniform(1,2);
a(n) = uniform (10,50);
l(n) = uniform(0.1,10);
u(n) = l(n) + uniform(0,12-l(n));

Variables x(n);
x.l(n) = uniform(l(n), u(n));
b = sum(n, x.l(n)*a(n));

Variables t(n), z(n), obj;
Equations defobjc, defobj, e1, e2(n), cone(n), conenlp(n);

defobjc..     sum(n, d(n)*t(n)) =E= obj;
defobj..      sum(n, d(n)/x(n)) =E= obj;
e1..          sum(n, a(n)*x(n)) =L= b;
e2(n)..       z(n) =E= sqrt(2);
cone(n)..     x(n) + t(n) =C= z(n);
conenlp(n)..  2*t(n)*x(n) =G= 2;

Model clp  /defobjc, e1, e2, cone/;
Model cnlp /defobjc, e1, conenlp/;
Model orig /defobj, e1/;

x.lo(n) = l(n);
x.up(n) = u(n);

Solve clp  min obj using lp;
Solve cnlp min obj using nlp;
Solve orig min obj using nlp;
```

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 more than 175 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         1
```

The first option specifies the maximum number of interior-point iterations, in this case 20. The seond option indicates a scaling option of 1, which is no scaling.

We remark that users can also use symbolic constants in place of numerical values. For example, for the scaling option users could use `MSK_SCALING_NONE` in place of the value 1. For a complete list of applicable symbolic constants, consult the MOSEK parameter list available online at `www.mosek.com`.

3.1 Memory Considerations for Nonlinear Problems

The GAMS *workfactor* option can be used to increase the amount of memory available to MOSEK. The general syntax is

(modelname).`workfactor` = *(value)*

with a default value of 1. See the section on "Using Solver Specific Options" for details. If GAMS/MOSEK runs out of memory, an error message is printed out:

```
 *** GAMS/MOSEK interface error.

            The size estimate for Hessian of Lagrangian is too small.
            Try to increase workfactor option from 1 to a larger value.
```

GAMS/MOSEK estimates the size of the Hessian as $5*(number\ of\ nonlinear\ variables)*(workfactor)$. Because of symmetry, the size of the Hessian is bounded by

$$H_d * (number\ of\ nonlinear\ variables)^2/2$$

where H_d detotes the density of the Hessian and $H_d \in [0,1]$. Therefore, one can choose the workfactor as:

$$workfactor = H_d * (number\ of\ nonlinear\ variables) * /(5*2)$$

Note that for a seperable model (see "Modeling Issues Involving Convex Programs"), the workfactor can in fact be reduced to 1/5.

4 Summary of MOSEK Options

4.1 General and Preprocessing Options

MSK_IPAR_CACHE_SIZE_L1	L1 cache size used
MSK_IPAR_CACHE_SIZE_L2	L2 cache size used
MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS	maximum number of optimizers during concurrent run
MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX	priority of dual simplex algorithm in concurrent run
MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX	priority of free simplex algorithm in concurrent run
MSK_IPAR_CONCURRENT_PRIORITY_INTPNT	priority of interior point algorithm in concurrent run
MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX	priority of primal simplex algorithm in concurrent run
MSK_IPAR_CPU_TYPE	specifies the CPU type
MSK_IPAR_INFEAS_REPORT_AUTO	

	switch for infeasibility report
MSK_IPAR_INFEAS_REPORT_LEVEL	output level for infeasibility report
MSK_IPAR_OPTIMIZER	optimizer selection
MSK_DPAR_OPTIMIZER_MAX_TIME	time limit
MSK_SPAR_PARAM_READ_FILE_NAME	name of a secondary MOSEK option file
MSK_IPAR_PRESOLVE_ELIMINATOR_USE	switch for free variable elimination
MSK_IPAR_PRESOLVE_ELIM_FILL	fill-in control during presolve
MSK_IPAR_PRESOLVE_LINDEP_USE	linear dependency check
MSK_IPAR_PRESOLVE_LINDEP_WORK_LIM	maximum work for finding linear dependencies
MSK_IPAR_PRESOLVE_USE	switch for presolve

4.2 Problem Data Options

MSK_IPAR_CHECK_CONVEXITY	level of convexity check for quadratic problems
MSK_DPAR_DATA_TOL_AIJ	zero tolerance for matrix coefficients
MSK_DPAR_DATA_TOL_AIJ_HUGE	error for large coefficients in matrix
MSK_DPAR_DATA_TOL_AIJ_LARGE	warning for large coefficients in matrix
MSK_DPAR_DATA_TOL_BOUND_INF	bound value for infinity
MSK_DPAR_DATA_TOL_BOUND_WRN	warning for large bounds
MSK_DPAR_DATA_TOL_CJ_LARGE	warning for large coefficients in objective
MSK_DPAR_DATA_TOL_C_HUGE	error for huge coefficients in objective
MSK_DPAR_DATA_TOL_QIJ	zero tolerance for Q matrix coefficients
MSK_DPAR_DATA_TOL_X	tolerance for fixed variables
MSK_DPAR_LOWER_OBJ_CUT	lower objective limit
MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH	upper objective limit threashold
MSK_DPAR_UPPER_OBJ_CUT	upper objective limit
MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH	lower objective limit threashold

4.3 Output Options

MSK_IPAR_LOG_BI
: output control for basis identification

MSK_IPAR_LOG_BI_FREQ
: frequency of log output of basis identification

MSK_IPAR_LOG_INTPNT
: output level of the interior-point optimizer

MSK_IPAR_LOG_MIO
: output level for mixed integer optimizer

MSK_IPAR_LOG_MIO_FREQ
: frequency of log output of mixed integer optimizer

MSK_IPAR_LOG_PRESOLVE
: output level for presolve

MSK_IPAR_LOG_SIM
: output level for simplex

MSK_IPAR_LOG_SIM_FREQ
: frequency of log output of simplex optimizer

MSK_IPAR_MAX_NUM_WARNINGS
: maximum number of warnings

MSK_IPAR_WARNING_LEVEL
: warning level

4.4 Interior Point Optimizer Options

MSK_IPAR_INTPNT_BASIS
: switch for basis identification

MSK_DPAR_INTPNT_CO_TOL_DFEAS
: dual feasibility tolerance for the conic interior-point optimizer

MSK_DPAR_INTPNT_CO_TOL_INFEAS
: infeasibility control for the conic interior-point optimizer

MSK_DPAR_INTPNT_CO_TOL_MU_RED
: relative complementarity tolerance for the conic interior-point optimizer

MSK_DPAR_INTPNT_CO_TOL_NEAR_REL
: termination tolerances for near optimal for the conic interior-point optimizer

MSK_DPAR_INTPNT_CO_TOL_PFEAS
: primal feasibility tolerance for the conic interior-point optimizer

MSK_DPAR_INTPNT_CO_TOL_REL_GAP
: relative optimality tolerance for the conic interior-point optimizer

MSK_IPAR_INTPNT_DIFF_STEP
: switch for different step sizes

MSK_IPAR_INTPNT_MAX_ITERATIONS
: iteration limit for the interior-point optimizer

MSK_IPAR_INTPNT_MAX_NUM_COR
: maximum number of correctors

MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS
: number of steps to be used by the iterative refinement

MSK_DPAR_INTPNT_NL_MERIT_BAL
: balance for complementarity and infeasibility

MSK_DPAR_INTPNT_NL_TOL_DFEAS
: dual feasibility tolerance for nonlinear problems

MSK_DPAR_INTPNT_NL_TOL_MU_RED

relative complementarity tolerance for nonlinear problems

MSK_DPAR_INTPNT_NL_TOL_PFEAS
primal feasibility tolerance for nonlinear problems

MSK_DPAR_INTPNT_NL_TOL_REL_GAP
relative optimality tolerance for nonlinear problems

MSK_IPAR_INTPNT_NUM_THREADS
number of threads for interior-point optimizer

MSK_IPAR_INTPNT_OFF_COL_TRH
offending column selection

MSK_IPAR_INTPNT_ORDER_METHOD
ordering strategy selection

MSK_IPAR_INTPNT_REGULARIZATION_USE
switch for regularization

MSK_IPAR_INTPNT_SCALING
scaling selection for interior-point optimizer

MSK_IPAR_INTPNT_SOLVE_FORM
solve primal or the dual problem with interior-point optimizer

MSK_IPAR_INTPNT_STARTING_POINT
starting point for interior-point optimizer

MSK_DPAR_INTPNT_TOL_DFEAS
dual feasibility tolerance

MSK_DPAR_INTPNT_TOL_DSAFE
initial dual starting control

MSK_DPAR_INTPNT_TOL_INFEAS
infeasibility control

MSK_DPAR_INTPNT_TOL_MU_RED
relative complementarity tolerance

MSK_DPAR_INTPNT_TOL_PATH
central path following for interior-point optimizer

MSK_DPAR_INTPNT_TOL_PFEAS
primal feasibility tolerance

MSK_DPAR_INTPNT_TOL_PSAFE
initial primal starting control

MSK_DPAR_INTPNT_TOL_REL_GAP
relative optimality tolerance

MSK_DPAR_INTPNT_TOL_REL_STEP
relative step size to boundary

USE_BASIS_EST
use MOSEK basis estimation in case of an interior solution

4.5 Simplex Optimizer and Basis Identification Options

MSK_IPAR_BI_CLEAN_OPTIMIZER
simplex optimizer section after basis identification

MSK_IPAR_BI_IGNORE_MAX_ITER
continues BI in case of iteration limit

MSK_IPAR_BI_IGNORE_NUM_ERROR
continues BI in case of numerical error

MSK_IPAR_BI_MAX_ITERATIONS
maximum number of simplex iterations after basis identification

MSK_IPAR_SIM_DUAL_CRASH
dual simplex crash

MSK_IPAR_SIM_DUAL_SELECTION

dual simplex pricing selection

MSK_IPAR_SIM_HOTSTART
: controls simplex hotstart

MSK_DPAR_SIM_LU_TOL_REL_PIV
: relative pivot tolerance for simplex and basis identification

MSK_IPAR_SIM_MAX_ITERATIONS
: simplex iteration limit

MSK_IPAR_SIM_MAX_NUM_SETBACKS
: maximum number of setbacks

MSK_IPAR_SIM_PRIMAL_CRASH
: primal simplex crash

MSK_IPAR_SIM_PRIMAL_SELECTION
: primal simplex pricing selection

MSK_IPAR_SIM_REFACTOR_FREQ
: refactorization frequency

MSK_IPAR_SIM_REFORMULATION
: controls if the simplex optimizers are allowed to reformulate

MSK_IPAR_SIM_SCALING
: scaling selection for simplex optimizer

MSK_IPAR_SIM_SCALING_METHOD
: controls how the problem is scaled before a simplex optimizer is used

MSK_IPAR_SIM_SOLVE_FORM
: solve primal or the dual problem with simplex optimizer

4.6 Mixed Integer Optimizer Options

MSK_IPAR_MIO_BRANCH_DIR
: control branching directions

MSK_IPAR_MIO_CONSTRUCT_SOL
: switch for mip start

MSK_IPAR_MIO_CUT_LEVEL_ROOT
: cut level control at root for mixed integer optimizer

MSK_IPAR_MIO_CUT_LEVEL_TREE
: cut level control in tree for mixed integer optimizer

MSK_IPAR_MIO_HEURISTIC_LEVEL
: heuristic control for mixed integer optimizer

MSK_DPAR_MIO_HEURISTIC_TIME
: time limit for heuristic search

MSK_IPAR_MIO_KEEP_BASIS
: switch for basis saving

MSK_IPAR_MIO_MAX_NUM_BRANCHES
: maximum number of branches

MSK_IPAR_MIO_MAX_NUM_RELAXS
: maximum number of relaxations solved

MSK_DPAR_MIO_MAX_TIME
: time limit for mixed integer optimizer

MSK_DPAR_MIO_MAX_TIME_APRX_OPT
: time limit before some relaxation

MSK_DPAR_MIO_NEAR_TOL_ABS_GAP
: termination criterion on absolute optimality tolerance

MSK_DPAR_MIO_NEAR_TOL_REL_GAP
: termination criterion on relative optimality tolerance

MSK_IPAR_MIO_NODE_OPTIMIZER

solver for the sub problems

MSK_IPAR_MIO_NODE_SELECTION
: node selection strategy

MSK_IPAR_MIO_PRESOLVE_AGGREGATE
: switch for aggregation during mixed integer presolve

MSK_IPAR_MIO_PRESOLVE_PROBING
: switch for probing

MSK_IPAR_MIO_PRESOLVE_USE
: switch for mixed integer presolve

MSK_DPAR_MIO_REL_ADD_CUT_LIMITED
: cuts factor

MSK_IPAR_MIO_ROOT_OPTIMIZER
: solver for the root problem

MSK_IPAR_MIO_STRONG_BRANCH
: strong branching control

MSK_DPAR_MIO_TOL_ABS_GAP
: absolute optimality tolerance in the mixed integer optimizer

MSK_DPAR_MIO_TOL_ABS_RELAX_INT
: absolute integrality tolerance

MSK_DPAR_MIO_TOL_FEAS
: feasibility tolerance for mixed integer solver

MSK_DPAR_MIO_TOL_REL_GAP
: relative optimality tolerance in the mixed integer optimizer

MSK_DPAR_MIO_TOL_REL_RELAX_INT
: relative integrality tolerance

MIPSTART
: use integer solution provided by user

5 Detailed Descriptions of MOSEK Options

MSK_IPAR_CACHE_SIZE_L1 *(integer)*

Controls the size of the L1 cache used by MOSEK.

(default = -1)

MSK_IPAR_CACHE_SIZE_L2 *(integer)*

Controls the size of the L2 cache used by MOSEK.

(default = -1)

MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS *(integer)*

(default = 2)

MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX *(integer)*

(default = 2)

MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX *(integer)*

(default = 3)

MSK_IPAR_CONCURRENT_PRIORITY_INTPNT *(integer)*

(default = 4)

MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX *(integer)*

(default = 1)

MSK_IPAR_CPU_TYPE (*string*)

This option specifies the CPU type.

- MSK_CPU_AMD_ATHLON
- MSK_CPU_AMD_OPTERON
- MSK_CPU_GENERIC
- MSK_CPU_INTEL_CORE2
- MSK_CPU_INTEL_P3
- MSK_CPU_INTEL_P4
- MSK_CPU_INTEL_PM
- MSK_CPU_POWERPC_G5
- MSK_CPU_UNKNOWN

MSK_IPAR_INFEAS_REPORT_AUTO (*integer*)

Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

(default = 0)

MSK_IPAR_INFEAS_REPORT_LEVEL (*integer*)

Controls the amount info presented in an infeasibility report. Higher values implies more information.

(default = 1)

MSK_IPAR_OPTIMIZER (*string*)

Controls which optimizer is used to optimize the task.

- MSK_OPTIMIZER_CONCURRENT
- MSK_OPTIMIZER_CONIC
- MSK_OPTIMIZER_DUAL_SIMPLEX
- MSK_OPTIMIZER_FREE
- MSK_OPTIMIZER_FREE_SIMPLEX
- MSK_OPTIMIZER_INTPNT
- MSK_OPTIMIZER_MIXED_INT
- MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX
- MSK_OPTIMIZER_PRIMAL_SIMPLEX
- MSK_OPTIMIZER_QCONE

MSK_DPAR_OPTIMIZER_MAX_TIME (*real*)

Maximum amount of time the optimizer is allowed to spend on the optimization. A negative number means infinity.

(default = GAMS ResLim)

MSK_SPAR_PARAM_READ_FILE_NAME (*string*)

The name of a secondary MOSEK option file that by the MOSEK option reader.

MSK_IPAR_PRESOLVE_ELIMINATOR_USE (*integer*)

Controls whether free or implied free variables are eliminated from the problem.

(default = 1)

MSK_IPAR_PRESOLVE_ELIM_FILL (*integer*)

Controls the maximum amount of fill-in that can be created during the eliminations 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_LINDEP_USE (*integer*)

Controls whether the linear constraints is checked for linear dependencies.

(default = 1)

MSK_IPAR_PRESOLVE_LINDEP_WORK_LIM (*integer*)

Is used to limit the work that can used to locate the linear dependencies. In general the higher value this parameter is given the less work can be used. However, a value of 0 means no limit on the amount work that can be used.

(default = 1)

MSK_IPAR_PRESOLVE_USE (*string*)

Controls whether presolve is performed.

- MSK_PRESOLVE_MODE_FREE
- MSK_PRESOLVE_MODE_OFF
- MSK_PRESOLVE_MODE_ON

MSK_IPAR_CHECK_CONVEXITY (*string*)

Specify the level of convexity check on quadratic problems.

- MSK_CHECK_CONVEXITY_FULL
- MSK_CHECK_CONVEXITY_NONE
- MSK_CHECK_CONVEXITY_SIMPLE

MSK_DPAR_DATA_TOL_AIJ (*real*)

Absolute zero tolerance for coefficients in the constraint matrix.

Range: [1.0e-16,1.0e-6]

(default = 1.0e-12)

MSK_DPAR_DATA_TOL_AIJ_HUGE (*real*)

An element in the constraint matrix which is larger than this value in absolute size causes an error.

DATA_TOL_BOUND_INF Any bound which in absolute value is greater than this parameter is considered infinite.

(default = 1.0e20)

MSK_DPAR_DATA_TOL_AIJ_LARGE (*real*)

A coefficient in the constraint matrix which is larger than this value in absolute size causes a warning message to be printed.

(default = 1.0e10)

MSK_DPAR_DATA_TOL_BOUND_INF (*real*)

(default = 1.0e16)

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 = 1.0e8)

MSK_DPAR_DATA_TOL_CJ_LARGE (*real*)

A coefficient in the objective which is larger than this value in absolute terms causes a warning message to be printed.

(default = 1.0e8)

MSK_DPAR_DATA_TOL_C_HUGE (*real*)

A coefficient in the objective which is larger than the value of this parameter in absolute terms is considered to be huge and generates an error.

(default = 1.0e16)

MSK_DPAR_DATA_TOL_QIJ (*real*)

Absolute zero tolerance for coefficients in the Q matrices.

(default = 1.0e-16)

MSK_DPAR_DATA_TOL_X (*real*)

Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and lower bound is considered identical.

(default = 1.0e-8)

MSK_DPAR_LOWER_OBJ_CUT (*real*)

If a feasible solution having and objective value outside, the interval `LOWER_OBJ_CUT`, `UPPER_OBJ_CUT`, then MOSEK is terminated.

(default = -1.0e30)

MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH (*real*)

If the lower objective cut (`LOWER_OBJ_CUT`) is less than `LOWER_OBJ_CUT_FINITE_TRH`, then the lower objective cut `LOWER_OBJ_CUT` is treated as infinity.

(default = -0.5e30)

MSK_DPAR_UPPER_OBJ_CUT (*real*)

If a feasible solution having and objective value outside, the interval `LOWER_OBJ_CUT`, `UPPER_OBJ_CUT`, then MOSEK is terminated.

(default = 1.0e30)

MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH (*real*)

If the upper objective cut (`UPPER_OBJ_CUT`) is greater than `UPPER_OBJ_CUT_FINITE_TRH`, then the upper objective cut `UPPER_OBJ_CUT` is treated as infinity.

(default = 0.5e30)

MSK_IPAR_LOG_BI (*integer*)

Controls the amount of output printed by the basis identification procedure.

(default = 4)

MSK_IPAR_LOG_BI_FREQ (*integer*)

Controls how frequent the optimizer outputs information about the basis identification is called.

(default = 2500)

MSK_IPAR_LOG_INTPNT (*integer*)

Controls the amount of output printed by the interior-point optimizer.

(default = 4)

MSK_IPAR_LOG_MIO (*integer*)

Controls the print level for the mixed integer optimizer.

(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_PRESOLVE (*integer*)

Controls amount of output printed by the presolve procedure.

(default = 1)

MSK_IPAR_LOG_SIM (*integer*)

Controls amount of output printed by the simplex optimizer.

(default = 4)

MSK_IPAR_LOG_SIM_FREQ (*integer*)

Controls how frequent the simplex optimizer outputs information about the optimization.

(default = 500)

MSK_IPAR_MAX_NUM_WARNINGS (*integer*)

Sets the maximum number of warnings.

(default = 10)

MSK_IPAR_WARNING_LEVEL (*integer*)

Warning level. A higher value implies more warnings.

(default = 1)

MSK_IPAR_INTPNT_BASIS (*string*)

Controls whether the interior-point optimizer also computes an optimal basis.

- MSK_BI_ALWAYS
- MSK_BI_IF_FEASIBLE
- MSK_BI_NEVER
- MSK_BI_NO_ERROR
- MSK_BI_OTHER

MSK_DPAR_INTPNT_CO_TOL_DFEAS (*real*)

Dual feasibility tolerance used by the conic interior-point optimizer.

Range: [0.0,1.0]

(default = 1.0e-8)

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 = 1.0e-8)

MSK_DPAR_INTPNT_CO_TOL_MU_RED (*real*)

Relative complementarity gap tolerance feasibility tolerance used by the conic interior-point optimizer.

Range: [0.0,1.0]

(default = 1.0e-8)

MSK_DPAR_INTPNT_CO_TOL_NEAR_REL (*real*)

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 = 100)

MSK_DPAR_INTPNT_CO_TOL_PFEAS (*real*)

Primal feasibility tolerance used by the conic interior-point optimizer.

Range: [0.0,1.0]

(default = 1.0e-8)

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 = 1.0e-8)

MSK_IPAR_INTPNT_DIFF_STEP (*integer*)

Controls whether different step sizes are allowed in the primal and dual space.

(default = 1)

MSK_IPAR_INTPNT_MAX_ITERATIONS (*integer*)

Sets 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_DPAR_INTPNT_NL_MERIT_BAL (*real*)

Controls if the complementarity and infeasibility is converging to zero at about equal rates.

Range: [0.0,0.99]

(default = 1.0e-4)

MSK_DPAR_INTPNT_NL_TOL_DFEAS (*real*)

Dual feasibility tolerance used when a nonlinear model is solved.

Range: [0.0,1.0]

(default = 1.0e-8)

MSK_DPAR_INTPNT_NL_TOL_MU_RED (*real*)

Relative complementarity gap tolerance used when a nonlinear model is solved..

Range: [0.0,1.0]

(default = 1.0e-12)

MSK_DPAR_INTPNT_NL_TOL_PFEAS (*real*)

Primal feasibility tolerance used when a nonlinear model is solved.

Range: [0.0,1.0]

(default = 1.0e-8)

MSK_DPAR_INTPNT_NL_TOL_REL_GAP (*real*)

Relative gap termination tolerance for nonlinear problems.

(default = 1.0e-6)

MSK_IPAR_INTPNT_NUM_THREADS (*integer*)

Controls the number of threads employed by the interior-point optimizer.

(default = 1)

MSK_IPAR_INTPNT_OFF_COL_TRH (*integer*)

Controls how many offending columns there are located in the Jacobian the constraint matrix. 0 means no offending columns will be detected. 1 means many offending columns will be detected. In general by increasing the number fewer offending columns will be detected.

(default = 40)

MSK_IPAR_INTPNT_ORDER_METHOD (*string*)

Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

- MSK_ORDER_METHOD_APPMINLOC1
- MSK_ORDER_METHOD_APPMINLOC2
- MSK_ORDER_METHOD_FREE
- MSK_ORDER_METHOD_GRAPHPAR1
- MSK_ORDER_METHOD_GRAPHPAR2
- MSK_ORDER_METHOD_NONE

MSK_IPAR_INTPNT_REGULARIZATION_USE (*integer*)

Controls whether regularization is allowed.

(default = 1)

MSK_IPAR_INTPNT_SCALING (*string*)

Controls how the problem is scaled before the interior-point optimizer is used.

- MSK_SCALING_AGGRESSIVE
- MSK_SCALING_FREE
- MSK_SCALING_MODERATE
- MSK_SCALING_NONE

MSK_IPAR_INTPNT_SOLVE_FORM (*string*)

Controls whether the primal or the dual problem is solved.

- MSK_SOLVE_DUAL

- MSK_SOLVE_FREE
- MSK_SOLVE_PRIMAL

MSK_IPAR_INTPNT_STARTING_POINT (*string*)

Selection of starting point used by the interior-point optimizer.

- MSK_STARTING_POINT_CONSTANT
- MSK_STARTING_POINT_FREE
- MSK_STARTING_POINT_GUESS
- MSK_STARTING_POINT_SATISFY_BOUNDS

MSK_DPAR_INTPNT_TOL_DFEAS (*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 (*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 (*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-8)

MSK_DPAR_INTPNT_TOL_MU_RED (*real*)

Relative complementarity gap tolerance

Range: [0.0,1.0]

(default = 1.0e-16)

MSK_DPAR_INTPNT_TOL_PATH (*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 (*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 (*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 (*real*)

Relative gap termination tolerance.

(default = 1.0e-8)

MSK_DPAR_INTPNT_TOL_REL_STEP (*real*)

Relative step size to the boundary for linear and quadratic optimization problems.

Range: [1.0e-4,0.999999]

(default = 0.9999)

USE_BASIS_EST (*integer*)

(default = 0)

MSK_IPAR_BI_CLEAN_OPTIMIZER (*string*)

Controls which simplex optimizer that is used in the clean up phase.

- MSK_OPTIMIZER_CONCURRENT
- MSK_OPTIMIZER_CONIC
- MSK_OPTIMIZER_DUAL_SIMPLEX
- MSK_OPTIMIZER_FREE
- MSK_OPTIMIZER_FREE_SIMPLEX
- MSK_OPTIMIZER_INTPNT
- MSK_OPTIMIZER_MIXED_INT
- MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX
- MSK_OPTIMIZER_PRIMAL_SIMPLEX
- MSK_OPTIMIZER_QCONE

MSK_IPAR_BI_IGNORE_MAX_ITER (*integer*)

If the parameter MSK_IPAR_INTPNT_BASIS has the value 2 and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value 1.

(default = 0)

MSK_IPAR_BI_IGNORE_NUM_ERROR (*integer*)

If the parameter MSK_IPAR_INTPNT_BASIS has the value 2 and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value 1.

(default = 0)

MSK_IPAR_BI_MAX_ITERATIONS (*integer*)

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

(default = 1000000)

MSK_IPAR_SIM_DUAL_CRASH (*integer*)

Controls whether crashing is performed in the dual simplex optimizer. In general if a basis consists of more than `(100*SIM_DUAL_CRASH)` percent fixed variables, then a crash will be performed.

(default = GAMS BRatio)

MSK_IPAR_SIM_DUAL_SELECTION (*string*)

Controls the choice of the incoming variable known as the selection strategy in the dual simplex optimizer.

- MSK_SIM_SELECTION_ASE

- MSK_SIM_SELECTION_DEVEX
- MSK_SIM_SELECTION_FREE
- MSK_SIM_SELECTION_FULL
- MSK_SIM_SELECTION_PARTIAL
- MSK_SIM_SELECTION_SE

MSK_IPAR_SIM_HOTSTART (*string*)

Controls whether the simplex optimizer will do hotstart if possible.

- MSK_SIM_HOTSTART_FREE
- MSK_SIM_HOTSTART_NONE
- MSK_SIM_HOTSTART_STATUS_KEYS

MSK_DPAR_SIM_LU_TOL_REL_PIV (*real*)

Relative pivot tolerance 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_IPAR_SIM_MAX_ITERATIONS (*integer*)

Maximum number of iterations that can used by a simplex optimizer.

(default = GAMS IterLim)

MSK_IPAR_SIM_MAX_NUM_SETBACKS (*integer*)

Controls how many setbacks that are allowed within a simplex optimizer. A setback 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_PRIMAL_CRASH (*integer*)

Controls whether crashing is performed in the primal simplex optimizer. In general if a basis consists of more than `(100*SIM_PRIMAL_CRASH)` percent fixed variables, then a crash will be performed.

(default = GAMS BRatio)

MSK_IPAR_SIM_PRIMAL_SELECTION (*string*)

Controls the choice of the incoming variable known as the selection strategy in the primal simplex optimizer.

- MSK_SIM_SELECTION_ASE
- MSK_SIM_SELECTION_DEVEX
- MSK_SIM_SELECTION_FREE
- MSK_SIM_SELECTION_FULL
- MSK_SIM_SELECTION_PARTIAL
- MSK_SIM_SELECTION_SE

MSK_IPAR_SIM_REFACTOR_FREQ (*integer*)

Controls how frequent the basis is refactorized. The value 0 means that the optimizer determines when the best point of refactorization is.

(default = 0)

MSK_IPAR_SIM_REFORMULATION (*string*)

- MSK_SIM_REFORMULATION_AGGRESSIVE
- MSK_SIM_REFORMULATION_FREE
- MSK_SIM_REFORMULATION_OFF
- MSK_SIM_REFORMULATION_ON

MSK_IPAR_SIM_SCALING (*string*)

Controls how the problem is scaled before a simplex optimizer is used.

- MSK_SCALING_AGGRESSIVE
- MSK_SCALING_FREE
- MSK_SCALING_MODERATE
- MSK_SCALING_NONE

MSK_IPAR_SIM_SCALING_METHOD (*string*)

- MSK_SCALING_METHOD_FREE
- MSK_SCALING_METHOD_POW2

MSK_IPAR_SIM_SOLVE_FORM (*string*)

Controls whether the primal or the dual problem is solved by the simplex optimizers.

- MSK_SOLVE_DUAL
- MSK_SOLVE_FREE
- MSK_SOLVE_PRIMAL

MSK_IPAR_MIO_BRANCH_DIR (*string*)

Controls whether the mixed integer optimizer is branching up or down by default.

- MSK_BRANCH_DIR_DOWN
- MSK_BRANCH_DIR_FREE
- MSK_BRANCH_DIR_UP

MSK_IPAR_MIO_CONSTRUCT_SOL (*integer*)

If set to 1 and all integer variables has been given a value for which a feasible MIP solution exists, then MOSEK generates an initial solution to the MIP by fixing all integer values and solving for the continues variables.

(default = 0)

MSK_IPAR_MIO_CUT_LEVEL_ROOT (*integer*)

Controls the cut level employed by the mixed integer optimizer. 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.

```
GUB cover              +2
Flow cover             +4
Lifting                +8
Plant location         +16
Disaggregation         +32
Knapsack cover         +64
Lattice                +128
Gomory                 +256
Coefficient reduction  +512
GCD                    +1024
Obj. integrality       +2048
```

(default = -1)

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.

(default = -1)

MSK_IPAR_MIO_HEURISTIC_LEVEL (*integer*)

Controls the heuristic employed by the mixed integer optimizer to locate an integer feasible solution. A value of zero means no heuristic is used. A large value than 0 means a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic to be used.

(default = -1)

MSK_DPAR_MIO_HEURISTIC_TIME (*real*)

Maximum time allowed to be used in the heuristic search for an optimal integer solution. A negative values implies that the optimizer decides the amount of time to be spend in the heuristic.

(default = -1.0)

MSK_IPAR_MIO_KEEP_BASIS (*integer*)

Controls whether the integer presolve keeps bases in memory. This speeds on the solution process at cost of bigger memory consumption.

(default = 1)

MSK_IPAR_MIO_MAX_NUM_BRANCHES (*integer*)

Maximum number branches allowed during the branch and bound search. A negative value means infinite.

(default = -1)

MSK_IPAR_MIO_MAX_NUM_RELAXS (*integer*)

Maximum number relaxations allowed during the branch and bound search. A negative value means infinite.

(default = -1)

MSK_DPAR_MIO_MAX_TIME (*real*)

This parameter limits the maximum time spend by the mixed integer optimizer. A negative number means infinity.

(default = -1.0)

MSK_DPAR_MIO_MAX_TIME_APRX_OPT (*real*)

Number of seconds spend by the mixed integer optimizer 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. The mixed integer optimizer is terminated when this tolerance is satisfied.

(default = GAMS OptCa)

MSK_DPAR_MIO_NEAR_TOL_REL_GAP (*real*)

Relaxed relative optimality tolerance employed by the mixed integer optimizer. The mixed integer optimizer is terminated when this tolerance is satisfied.

(default = GAMS OptCr)

MSK_IPAR_MIO_NODE_OPTIMIZER (*string*)

Controls which optimizer is employed at non root nodes in the mixed integer optimizer.

- MSK_OPTIMIZER_CONCURRENT
- MSK_OPTIMIZER_CONIC
- MSK_OPTIMIZER_DUAL_SIMPLEX
- MSK_OPTIMIZER_FREE
- MSK_OPTIMIZER_FREE_SIMPLEX
- MSK_OPTIMIZER_INTPNT
- MSK_OPTIMIZER_MIXED_INT
- MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX
- MSK_OPTIMIZER_PRIMAL_SIMPLEX
- MSK_OPTIMIZER_QCONE

MSK_IPAR_MIO_NODE_SELECTION (*string*)

Controls the node selection strategy employed by the mixed integer optimizer.

- MSK_MIO_NODE_SELECTION_BEST
- MSK_MIO_NODE_SELECTION_FIRST
- MSK_MIO_NODE_SELECTION_FREE
- MSK_MIO_NODE_SELECTION_HYBRID
- MSK_MIO_NODE_SELECTION_PSEUDO
- MSK_MIO_NODE_SELECTION_WORST

MSK_IPAR_MIO_PRESOLVE_AGGREGATE (*integer*)

Controls whether the presolve used by the mixed integer optimizer tries to aggregate the constraints.

(default = 1)

MSK_IPAR_MIO_PRESOLVE_PROBING (*integer*)

Controls whether the mixed integer presolve performs probing. Probing can be very time consuming.

(default = 1)

MSK_IPAR_MIO_PRESOLVE_USE (*integer*)

Controls whether presolve is performed by the mixed integer optimizer.

(default = 1)

MSK_DPAR_MIO_REL_ADD_CUT_LIMITED (*real*)

Controls how many cuts the mixed integer optimizer is allowed to add to the problem. The mixed integer optimizer is allowed to `MIO_REL_ADD_CUT_LIMITED`$*m$ w cuts, where m is the number constraints in the problem.

Range: [0.0,2.0]

(default = 0.75)

MSK_IPAR_MIO_ROOT_OPTIMIZER (*string*)

Controls which optimizer is employed at the root node in the mixed integer optimizer.

- MSK_OPTIMIZER_CONCURRENT
- MSK_OPTIMIZER_CONIC
- MSK_OPTIMIZER_DUAL_SIMPLEX
- MSK_OPTIMIZER_FREE
- MSK_OPTIMIZER_FREE_SIMPLEX
- MSK_OPTIMIZER_INTPNT

- MSK_OPTIMIZER_MIXED_INT
- MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX
- MSK_OPTIMIZER_PRIMAL_SIMPLEX
- MSK_OPTIMIZER_QCONE

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_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, i.e. 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 = 1.0e-4)

MSK_DPAR_MIO_TOL_REL_RELAX_INT (*real*)

Relative relaxation tolerance of the integer constraints, i.e. 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)

MIPSTART (*integer*)

(default = 0)

0 No mipstart

1 Mipstart with discrete variables only. Solve fixed problem first

2 Mipstart with all variables, including continuous

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 `trnsport.gms` from the GAMS Model Library:

```
Interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Presolve   - time                   : 0.00
Presolve   - Stk. size (kb)         : 0
Eliminator - tries                  : 0                 time                   : 0.00
Eliminator - elim's                 : 0
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated.
Matrix reordering started.
Local matrix reordering started.
Local matrix reordering terminated.
Matrix reordering terminated.
Optimizer  - threads                : 1
Optimizer  - solved problem         : the primal
Optimizer  - constraints            : 5                 variables              : 11
Factor     - setup time             : 0.00              order time             : 0.00
Factor     - GP order used          : no                GP order time          : 0.00
Factor     - nonzeros before factor : 11                after factor           : 13
Factor     - offending columns      : 0                 flops                  : 2.60e+01
```

The first part gives information about the presolve (if used). The main log follows:

```
ITE PFEAS    DFEAS    KAP/TAU  POBJ               DOBJ               MU       TIME
0   6.0e+02  1.0e+00  1.0e+00  1.053000000e+00    0.000000000e+00    1.2e+01  0.00
1   5.9e+02  1.1e+00  1.0e+00  3.063646498e+00    5.682895191e+00    3.0e+01  0.00
2   4.6e+01  8.6e-02  9.8e+00  3.641071165e+01    4.750801284e+01    2.3e+00  0.00
3   8.7e-01  1.6e-03  1.7e+01  1.545771936e+02    1.719072826e+02    4.4e-02  0.00
4   8.1e-02  1.5e-04  8.8e-01  1.543678291e+02    1.552521470e+02    4.1e-03  0.00
5   1.3e-02  2.4e-05  1.3e-01  1.537617961e+02    1.538941635e+02    6.4e-04  0.00
6   1.3e-03  2.4e-06  1.1e-02  1.536766256e+02    1.536876562e+02    6.6e-05  0.00
7   1.6e-07  3.1e-10  1.2e-06  1.536750013e+02    1.536750025e+02    8.4e-09  0.00
Basis identification started.
Primal basis identification phase started.
ITER      TIME
1         0.00
Primal basis identification phase terminated. Time: 0.00
Dual basis identification phase started.
ITER      TIME
0         0.00
Dual basis identification phase terminated. Time: 0.00
Basis identification terminated. Time: 0.00
Interior-point optimizer terminated. CPU Time: 0.00. Real Time: 0.00.

Interior-point solution
Problem status  : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal - objective: 1.5367500132e+02   eq. infeas.: 5.61e-06 max bound infeas.: 0.00e+00 cone infeas.:
Dual   - objective: 1.5367500249e+02   eq. infeas.: 1.06e-08 max bound infeas.: 0.00e+00 cone infeas.:

Basic solution
Problem status  : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal - objective: 1.5367500000e+02   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
```

```
Dual   - objective: 1.5367500000e+02   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00

Optimizer                 - real time  : 0.00       cpu time: 0.00
  Interior-point          - iterations : 7          cpu time: 0.00
    Basis identification  -                         cpu time: 0.00
      Primal              - iterations : 1          cpu time: 0.00
      Dual                - iterations : 0          cpu time: 0.00
      Clean               - iterations : 0          cpu time: 0.00
  Simplex                 -                         cpu time: 0.00
    Primal simplex        - iterations : 0          cpu time: 0.00
    Dual simplex          - iterations : 0          cpu time: 0.00
  Mixed integer           - relaxations: 0          cpu time: 0.00
```

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:

Field	**Description**
ITE	The number of the current iteration.
PFEAS	Primal feasibility.
DFEAS	Dual feasibility.
KAP/TAU	This measure should converge to zero if the problem has a primal/dual optimal solution. Whereas it should converge to infinity when the problem is (strictly) primal or dual infeasible. In the case the measure is converging towards a positive but bounded constant then the problem is usually ill-posed.
POBJ	Current objective function value of primal problem.
DOBJ	Current objective function value of dual problem.
MU	Relative complementary gap.
TIME	Current elapsed resource time in seconds.

6.2 Log Using the Simplex Optimizer

Below is a log output running the model `trnsport.gms` from the GAMS model library using the MOSEK simplex optimizer.

```
Reading parameter(s) from "mosek.opt"
>>  MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_DUAL_SIMPLEX
Simplex optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Presolve   - time                   : 0.00
Presolve   - Stk. size (kb)         : 0
Eliminator - tries                  : 0                 time                   : 0.00
Eliminator - elim's                 : 0
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated.
Dual simplex optimizer started.
Dual simplex optimizer setup started.
Dual simplex optimizer setup terminated.
Optimizer  - solved problem         : the primal
Optimizer  - constraints            : 5                 variables              : 6
Optimizer  - hotstart               : no
```

```
ITER      DEGITER%  FEAS                  DOBJ                  TIME(s)
0         0.00      0.0000000000e+00      0.0000000000e+00      0.00
3         0.00      0.0000000000e+00      1.5367500000e+02      0.00
Dual simplex optimizer terminated.
Simplex optimizer terminated.
Basic solution
Problem status  : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal - objective: 1.5367500000e+02   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
Dual   - objective: 1.5367500000e+02   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
```

The fields in the main MOSEK log output are:

Field	Description
ITER	Current number of iterations.
DEGITER%	Current percentage of degenerate iterations.
FEAS	Current (primal or dual) infeasibility.
D/POBJ	Current dual or primal objective
TIME	Current elapsed resource time in seconds.

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.

```
Mixed integer optimizer started.
BRANCHES RELAXS   ACT_NDS  BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(%)  TIME
0        1        0        1.6000000000e+01     0.0000000000e+00     100.00      0.1
0        1        0        4.0000000000e+00     0.0000000000e+00     100.00      0.1
128      250      5        4.0000000000e+00     0.0000000000e+00     100.00      0.3
167      502      6        4.0000000000e+00     0.0000000000e+00     100.00      0.7
241      758      65       4.0000000000e+00     0.0000000000e+00     100.00      0.9
200      809      83       4.0000000000e+00     1.3333333333e-01     96.67       0.9
A near optimal solution satisfying the absolute gap tolerance of 3.90e+00 has been located.


Objective of best integer solution : 4.00000000e+00
Number of branches                 : 267
Number of relaxations solved       : 810
Number of interior point iterations: 0
Number of simplex iterations       : 10521
Mixed integer optimizer terminated. Time: 0.95
```

The fields in the main MOSEK log output are:

Field	Description
BRANCHES	Current number of branches in tree.
RELAXS	Current number of nodes in branch and bound tree.
ACT_NDS	Current number of active nodes.
BEST_INT_OBJ.	Current best integer solution
BEST_RELAX_OBJ	Current best relaxed solution.
REL_GAP(%)	Relative gap between current BEST_INT_OBJ. and BEST_RELAX_OBJ.
TIME	Current elapsed resource time in seconds.

The log then gives information about solving the model with discrete variables fixed in order to determine marginals. 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.
Presolve   - time                 : 0.00
Presolve   - Stk. size (kb)       : 12
Eliminator - tries                : 0                time                  : 0.00
Eliminator - elim's               : 0
Lin. dep.  - tries                : 1                time                  : 0.00
Lin. dep.  - number               : 0
Presolve terminated.
Interior-point optimizer terminated. CPU Time: 0.00. Real Time: 0.00.

Interior-point solution
Problem status  : PRIMAL_FEASIBLE
Solution status : PRIMAL_FEASIBLE
Primal - objective: 4.0000000000e+00   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e
Dual   - objective: -8.0000000000e+00  eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e

Basic solution
Problem status  : PRIMAL_FEASIBLE
Solution status : PRIMAL_FEASIBLE
Primal - objective: 4.0000000000e+00   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
Dual   - objective: -8.0000000000e+00  eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00

Optimizer                - real time  : 1.35       cpu time: 0.95
  Interior-point         - iterations : 0          cpu time: 0.00
    Basis identification -                         cpu time: 0.00
      Primal             - iterations : 0          cpu time: 0.00
      Dual               - iterations : 0          cpu time: 0.00
      Clean              - iterations : 0          cpu time: 0.00
  Simplex                -                         cpu time: 0.00
    Primal simplex       - iterations : 0          cpu time: 0.00
    Dual simplex         - iterations : 0          cpu time: 0.00
  Mixed integer          - relaxations: 810        cpu time: 0.95
```

NLPEC

Contents

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 μ 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:

```
gams nash MPEC=nlpec MCP=nlpec
```

You can use NLPEC with its default strategy and formulation, but most users will want to use an options file (Section 2.4) after reading about the different types of reformulations possible (Section 3). In addition, an understanding of the architecture of NLPEC (Section 5) 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 4.1) mentioned throughout this section.

The original MPEC model is given as:

$$\min_{x\in\mathbf{R}^n, y\in\mathbf{R}^m} f(x,y) \tag{3.1}$$

subject to the constraints

$$g(x,y) \leq 0 \tag{3.2}$$

and

$$y \quad \text{solves} \quad \text{MCP}(h(x,\cdot),\mathbf{B}). \tag{3.3}$$

In most of the reformulations, the objective function (**??**) 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 (**??**) 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 (**??**), where (**??**) signifies that $y \in \mathbf{R}^m$ 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 (**??**) if for each i at least one of the following holds:

$$\begin{array}{c} h_i(x,y) = 0 \\ h_i(x,y) > 0,\ y_i = a_i; \\ h_i(x,y) < 0,\ y_i = b_i. \end{array} \tag{3.4}$$

As a special case of (**??**), consider the case where $a = 0$ and $b = +\infty$. Since y_i can never be $+\infty$ at a solution, (**??**) simplifies to the nonlinear complementarity problem (NCP):

$$0 \leq h_i(x,y), 0 \leq y_i \quad \text{and} \quad y_i h_i(x,y) = 0, i = 1,\ldots,m \tag{3.5}$$

namely that h and y are nonnegative vectors with h perpendicular to y. This motivates our shorthand for (**??**), the "perp to" symbol $\perp$:

$$h_i(x,y) \quad \perp \quad y_i \in [a_i,b_i] \tag{3.6}$$

The different ways to force (**??**) 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:

$$\begin{array}{cc} \min\limits_{x_1,x_2,y_1,y_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & y_1 - y_2 + 1 \leq x_1 \perp y_1 \geq 0 \\ & x_2 + y_2 \perp y_2 \in [-1,1] \end{array}$$

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:

```
$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.y1, h2.y2 /;
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 h_2 and not a constraint $h_2(x, y) = 0$, we use the GAMS syntax `=N=` to ensure this is clear here. Since the relationships satisfied by h_1 and h_2 are determined by the bounds, `=G=` could also be replaced by `=N=` in `h1`.

In describing the various reformulations for (**??**), 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:

$$\mathbf{B} := \{y = (y_\mathcal{F}, y_\mathcal{L}, y_\mathcal{U}, y_\mathcal{B}) \mid a_\mathcal{L} \le y_\mathcal{L},\ y_\mathcal{U} \le b_\mathcal{U},\ a_\mathcal{B} \le y_\mathcal{B} \le b_\mathcal{B}\}.$$

We will assume (without loss of generality) that $a_\mathcal{B} < b_\mathcal{B}$. If $a_i = b_i$ then (**??**) 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, $\mathcal{L} = \{1\}$, $\mathcal{U} = \emptyset$ and $\mathcal{B} = \{2\}$.

3.1 Product reformulations

Product reformulations all involve products of y_i with h_i, or products of y_i with some auxiliary or slack variables that are set equal to h_i. The underlying point is that the constraints (**??**) are entirely equivalent to the following system of equalities and inequalities:

$$\begin{array}{c} w_\mathcal{L} = h_\mathcal{L}(x,y),\ a_\mathcal{L} \le y_\mathcal{L},\ w_\mathcal{L} \ge 0 \quad \text{and} \quad (y_\mathcal{L} - a_\mathcal{L})^T w_\mathcal{L} = 0 \\ v_\mathcal{U} = -h_\mathcal{U}(x,y),\ y_\mathcal{U} \le b_\mathcal{U},\ v_\mathcal{U} \ge 0 \quad \text{and} \quad (b_\mathcal{U} - y_\mathcal{U})^T v_\mathcal{U} = 0 \\ w_\mathcal{B} - v_\mathcal{B} = h_\mathcal{B}(x,y),\ a_\mathcal{B} \le y_\mathcal{B} \le b_\mathcal{B},\ w_\mathcal{B} \ge 0,\ v_\mathcal{B} \ge 0 \\ (y_\mathcal{B} - a_\mathcal{B})^T w_\mathcal{B} = 0,\ (b_\mathcal{B} - y_\mathcal{B})^T v_\mathcal{B} = 0. \end{array} \tag{3.7}$$

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 (**??**) above, or broken out into individual products (one for each $i \in \mathcal{L} \cup \mathcal{U}$, two for each $i \in \mathcal{B}$). For example, the complementarity constraints associated with lower bounded variables involve nonnegativity of $w_{\mathcal{L}}$, $y_{\mathcal{L}} \geq a_{\mathcal{L}}$ and either of the following alternatives:

$$(y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = \sum (i \in \mathcal{L}(y_i - a_i)w_i = 0$$

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 (**??**) involve nonnegative terms, we can set the inner products equal to zero or set them ≤ 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:

$$\begin{array}{cc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 \leq \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1] \\ & (y_2 + 1)w_2 \leq \mu,\ (1 - y_2)v_2 \leq \mu \end{array} \tag{3.8}$$

By default, a single model is generated with the value μ 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 (**??**) above are always set equal to (or $\leq$) a scalar μ instead of zero. By default μ is zero, but options exist to start μ 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 μ (e.g. `FinalMu 0`). If the following lines are added to the option file

```
initmu 1.0
numsolves 4
```

then five consecutive solves of the nonlinear program (**??**) are performed, the first one using $\mu = 1$ and each subsequent solve dividing μ 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
numsolves 4
updatefac 0.1
finalmu 1e-6
```

produces 6 nonlinear programs of the form

$$\begin{array}{cc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1+x_2 \\ \text{subject to} & x_1^2+x_2^2 \le 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \ge 0, y_1 \ge 0 \\ & w_1 y_1 = \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \ge 0, y_2 \in [-1,1],\ (y_2+1)w_2 = \mu,\ (y_2-1)v_2 = \mu \end{array}$$

for values of $\mu = 1, 0.1, 0.01, 0.001, 0.0001$ and $1e-6$.

3.1.1 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 (??). For a simpler illustrative example the NCP constraints (??) are equivalent to the constraints:

$$w_i = h_i(x,y), 0 \le w_i, 0 \le y_i \quad \text{and} \quad y_i w_i = 0, i = 1,\ldots,m$$

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 (??). Since this is the default value of the option `slack`, the example (??) 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:

$$h_i \quad \perp \quad a_i \le y_i \le b_i \iff$$

$$a_i \le y_i \le b_i,\ (y_i - a_i)h_i \le \mu,\ (y_i - b_i)h_i \le \mu \tag{3.9}$$

Note that unlike the inner products in Section 3.1, we can expect that one of the inequalities in (??) 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

$$\begin{array}{cc} \min\limits_{x_1,x_2,y_1,y_2} & x_1+x_2 \\ \text{subject to} & x_1^2+x_2^2 \le 1 \\ & x_1 - y_1 + y_2 - 1 \ge 0, y_1 \ge 0 \\ & (x_1 - y_1 + y_2 - 1)y_1 = \mu \\ & y_2 \in [-1,1],\ (y_2+1)(x_2+y_2) \le \mu,\ (y_2-1)(x_2+y_2) \le \mu \end{array}$$

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:

$$h_i(x,y) \quad \perp \quad a_i \le y_i \le b_i \iff a_i \le y_i \le b_i,\ w_i = h_i(x,y),\ (y_i - a_i)w_i \le \mu,\ (y_i - b_i)w_i \le \mu$$

Note that the slack variable w 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

```
reftype mult
slack positive one
```

sets the option `slack positive` for the singly bounded variables and the option `slack one` for the doubly bounded variables resulting in the model

$$\begin{array}{cc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 = \mu_1 \\ & w_2 = x_2 + y_2,\ y_2 \in [-1,1],\ (y_2+1)w_2 \leq \mu_2,\ (y_2-1)w_2 \leq \mu_2 \end{array}$$

Additional options such as

```
initmu 1.0 3.0
numsolves 2
updatefac 0.1 0.2
```

allow the values of μ for the singly and doubly bounded variables to be controlled separately. In this case μ_1 takes on values of 1, 0.1 and 0.01, while μ_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 (**??**) 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 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 h_i into its positive and negative parts (but see Section 3.2.1 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 μ, and approach the true NCP-function as the smoothing parameter approaches zero. For example, the Fischer-Burmeister function includes a perturbation μ that guarantees differentiability:

$$\phi_{FB}(r,s) := \sqrt{r^2 + s^2 + 2\mu} - (r+s). \tag{3.10}$$

You can choose these particular NCP functions using option `RefType min|FB|fFB`. The difference between the last two is that `RefType FB` writes out GAMS code to compute the function ϕ_{FB}, while `RefType fFB` makes use of a GAMS intrinsic function NCPFB(r,s,mu) that computes ϕ_{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

```
reftype fFB
slack free
initmu 1e-2
```

generates the reformulation

$$\begin{array}{cc}\min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1+x_2\\ \text{subject to} & x_1^2+x_2^2\le 1\\ & w_1 = x_1 - y_1 + y_2 - 1\\ & \phi_{FB}(w_1,y_1,\mu)=0\\ & w_2 - v_2 = x_2 + y_2\\ & \phi_{FB}(y_2+1,w_2,\mu)=0,\ \phi_{FB}(1-y_2,v_2,\mu)=0\end{array}$$

with a value of $\mu = 0.01$. Following a path of solutions for decreasing values of μ 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. $h_i \ge 0, \phi(h_i, y_i - a_i) = 0$. 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

```
reftype min
slack positive
NCPBounds function
```

generates the reformulation

$$\begin{array}{cc}\min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1+x_2\\ \text{subject to} & x_1^2+x_2^2\le 1\\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \ge 0\\ & \min(w_1,y_1)=\mu\\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \ge 0\\ & \min(y_2+1,w_2)=\mu,\ \min(1-y_2,v_2)=\mu\end{array}$$

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 μ. For this reason, the model is constructed as a `dnlp` model (instead of an `nlp` model) in GAMS.

A smoothed version of the min function was proposed by Chen & Mangasarian:

$$\phi_{CM}(r,s) := r - \mu \log(1 + \exp((r-s)/\mu)). \tag{3.11}$$

This function is not symmetric in its two arguments, so $\phi_{CM}(r,s) \ne \phi_{CM}(s,r)$. For this reason, we distinguish between the two cases. Unlike the Fischer-Burmeister function ϕ_{FB}, ϕ_{CM} is not defined in the limit (i.e. for $\mu = 0$) if you use GAMS code to compute it. However, the GAMS intrinsic NCPCM(r,s,mu) handles this limit case internally. The option `RefType CMxf|CMfx|fCMxf|fCMfx` chooses a reformulation based on the function ϕ_{CM}. Again, the last two choices use the GAMS intrinsic function.

3.2.1 Doubly bounded variables

Like the mult reformulation (**??**), 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:

$$h_i \quad \perp \quad a_i \le y_i \le b_i \iff$$

$$\phi_{FB}(y_i - a_i, \phi_{FB}(b_i - y_i, -h_i)) = 0 \tag{3.12}$$

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 treat these by moving them into the objective function with a penalty parameter $1/\mu$: as μ goes to zero, the relative weight placed on complementarity increases. Ignoring the NLP constraints, we can rewrite the original MPEC problem as

$$\min_{x \in \mathbf{R}^n, y \in \mathbf{R}^m} f(x,y) + \frac{1}{\mu}((y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} + (b_{\mathcal{U}} - y_{\mathcal{U}})^T v_{\mathcal{U}} + (y_{\mathcal{B}} - a_{\mathcal{B}})^T w_{\mathcal{B}} + (b_{\mathcal{B}} - y_{\mathcal{B}})^T v_{\mathcal{B}}) \tag{3.13}$$

subject to the constraints

$$\begin{array}{c} g(x,y) \le 0 \\ w_{\mathcal{L}} = h_{\mathcal{L}}(x,y),\ a_{\mathcal{L}} \le y_{\mathcal{L}},\ w_{\mathcal{L}} \ge 0 \\ v_{\mathcal{U}} = -h_{\mathcal{U}}(x,y),\ y_{\mathcal{U}} \le b_{\mathcal{U}},\ v_{\mathcal{U}} \ge 0 \\ w_{\mathcal{B}} - v_{\mathcal{B}} = h_{\mathcal{B}}(x,y)\ a_{\mathcal{B}} \le y_{\mathcal{B}} \le b_{\mathcal{B}},\ w_{\mathcal{B}} \ge 0, v_{\mathcal{B}} \ge 0 \end{array} \tag{3.14}$$

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:

```
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:

$$\begin{array}{cc} \min\limits_{x_1,x_2,y_1,y_2,w_2,v_2} & x_1 + x_2 + 1/\mu_1 y_1(x_1 - y_1 + y_2 - 1) \\ \text{subject to} & x_1^2 + x_2^2 \le 1 \\ & x_1 - y_1 + y_2 - 1 \ge 0, y_1 \ge 0 \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \ge 0, y_2 \in [-1,1] \\ & (y_2+1)w_2 \le \mu_2,\ (1-y_2)v_2 \le \mu_2 \end{array}$$

The penalty parameter μ_1 is controlled separately from the doubly bounded constraint parameter μ_2. For consistency with other options, the penalty parameter in the objective is $1/\mu$ meaning that as μ_1 tends to zero, the penalty increases. The option `initmu` has only one value, so both the singly and doubly bounded μ 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 $\mathcal{L} \cup \mathcal{U}$ and another for the doubly-bounded variables in $\mathcal{B}$. 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

```
RefType  fCMxf
RefType  *  fBill
```

first sets the `RefType` to `fCMxf` for all variable types, and then resets the `RefType` to `fBill` for doubly-bounded variables.

Option	Description	Default
`reftype`	Determines the type of reformulation used - see Section 3 for details. Our notation and descriptions are taken from a special case of the MPEC, the NCP: find $x \geq 0, f(x) \geq 0, x^T f(x) = 0$. `mult` inner-product reformulation $x^T f = 0$ (Section 3.1) `min` NCP-function $\min(x, f)$ (Section 1.1.3) `CMxf` Chen-Mangasarian NCP-function $\phi_{CM}(x, f) := x - \mu \log(1 + \exp((x - f)/\mu))$, written explicitly in GAMS code (Section 1.1.3) `CMfx` Chen-Mangasarian NCP-function $\phi_{CM}(f, x) := f - \mu \log(1 + \exp((f - x)/\mu))$, written explicitly in GAMS code (Section 1.1.3) `fCMxf` Chen-Mangasarian NCP-function $\phi_{CM}(x, f) := x - \mu \log(1 + \exp((x - f)/\mu))$, using GAMS intrinsic NCPCM(x,f,μ) (Section 1.1.3) `fCMfx` Chen-Mangasarian NCP-function $\phi_{CM}(f, x) := f - \mu \log(1 + \exp((f - x)/\mu))$, using GAMS intrinsic NCPCM(f,x,μ) (Section 1.1.3) `FB` Fischer-Burmeister NCP-function $\phi_{FB}(x, f) := \sqrt{x^2 + f^2 + 2\mu} - (x + f)$, written explicitly in GAMS code (Section 1.1.3) `fFB` Fischer-Burmeister NCP-function $\phi_{FB}(x, f) := \sqrt{x^2 + f^2 + 2\mu} - (x + f)$, using GAMS intrinsic NCPFB(x,f,μ) (Section 1.1.3) `Bill` Billups function for doubly-bounded variables, written explicitly in GAMS code (Section 3.2.1) `fBill` Billups function for doubly-bounded variables, using GAMS intrinsic NCPFB(x,f,μ) (Section 3.2.1) `penalty` Penalization of non-complementarity in objective function (Section 3.3)	mult/mult
`slack`	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` nonnegative slacks will be used `one` one free slack will be used for each h_i in the doubly-bounded case.	positive/positive

`constraint`	Determines 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.
	`equality`
	`inequality`
		equality/equality

`aggregate`	Determines 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 $w_i^T y_i = 0, \forall i$. `none` Use no aggregation `partial` Aggregate terms in $\mathcal{L} \cup \mathcal{U}$ separately from those in $\mathcal{B}$ `full` Use maximum aggregation possible	none/none
`NCPBounds`	Determines which of the two arguments to an NCP function $\phi(r, s)$ are explicitly constrained to be nonnegative (see Section 1.1.3). 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 constraints `function` Explicit constraint for function argument `variable` Explicit constraint for variable argument `all` Explicit constraints for function and variable arguments	none/none

4.2 General Options

Option	Description	Default
`allsolves`	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.	no
`finalmu`	Final value of the parameter μ. If specified, an extra solve is carried out with μ set to this value. Can be set independently for singly and doubly bounded variables.	none
`initmu`	Initial value of the parameter μ. A single solve of the nonlinear program is carried out for this value. Note that μ must be positive for some settings of `reftype`, e.g. `penalty`. Can be set independently for singly and doubly bounded variables.	0.0
`initslo`	The lower bound for any artificials that are added.	0
`initsup`	The upper bound for any artificials that are added.	inf
`nocheck`	Turns off the consistency checks for the reformulation options (see Section 4.1).	off
`numsolves`	Number of extra solves carried out in a loop. This should be set in conjunction with the `updatefac` option.	0
`subsolver`	Selects NLP or DNLP subsolver to run. If no subsolver is chosen, the usual procedure for setting the solver is used.	auto
`subsolveropt`	Selects an options file for use with the NLP or DNLP subsolver.	0
`testtol`	Zero tolerance used for checking the complementarity gap of the proposed solution to the MPEC.	1e-5
`updatefac`	The factor that multiplies μ before each of the extra solves triggered by the `numsolves` option. Can be set independently for singly and doubly bounded variables.	0.1

4.3 The Outdated `equreform` Option

In the early versions of NLPEC the only way to set the reform type was via the `equreform` option. Each valid `equreform` value represented a preselected combination of the options from Section 4.1. This made it difficult to experiment with combinations not preselected, so the options in Section 4.1 were added. Be default, the `equreform` option has value 0 and is not used. To get the old behavior, set `equreform` to a positive value - this will force the options in Section 4.1 to be ignored. The general options in Section 4.2 are used no matter how the reformulation type is selected - via `RefType` or `equreform`.

Option	Description	Default
`equreform`	Old way to set the type of reformulation used.	0

The values allowed for `equreform` and their implications are given below.

	L/U				**B**			
equref	**reftype**	**sign**	**slacks**	**free-y**	**reftype**	**sign**	**slacks**	**free-y**
1	$<,>_i$	$=\mu$	bnd		$<,>_i$	$=\mu$	bnd	
2	$<,>_i$	$<=\mu$	bnd		$<,>_i$	$<=\mu$	bnd	
3	$<,>_i$	$=\mu$	bnd		Scholtes	$<=\mu$	one	
4	$<,>_{L+U+B}$	$=\mu$	bnd		$<,>_{L+U+B}$	$=\mu$	bnd	
5	$<,>_{L+U+B}$	$=\mu$	none		$<,>_{L+U+B}$	$=\mu$	bnd	
6	$<,>_{L+U}$	$=\mu$	none		Scholtes	$<=\mu$	one	
7	$<,>_{L+U}$	$<=\mu$	none		Scholtes	$<=\mu$	none	
8	$<,>_i$	$=\mu$	none		Scholtes	$<=\mu$	none	
9	$<,>_{\text{obj}}$		bnd		$<,>_{\text{obj}}$		bnd	
10	$<,>_{\text{obj}}$		none		$<,>_{\text{obj}}$		bnd	
11	$<,>_i$	$=\mu$	none		$<,>_i$	$=\mu$	bnd	
12	F-B	$=0$	none	free	F-B	$=0$	free	free
13	F-B	$=0$	none	free	Billups	$=0$	none	free
14	min	$=\mu$	free	free	min	$=\mu$	free	free
15	min	$<=\mu$	bnd		min	$<=\mu$	bnd	
16	C-M(x,f)	$=0$	free	free	C-M(x,f)	$=0$	free	free
17	C-M(x,f)	$=0$	bnd		C-M(x,f)	$=0$	bnd	
18	$<,>_{L,U}$	$<=\mu$	none		$<,>_B$	$<=\mu$	bnd	
19	$<,>_i$	$<=\mu$	none		$<,>_i$	$<=\mu$	bnd	
20	$<,>_{L,U}$	$<=\mu$	bnd		$<,>_B$	$<=\mu$	bnd	
21	$<,>_{L+U+B}$	$<=\mu$	bnd		$<,>_{L+U+B}$	$<=\mu$	bnd	
22	F-B	$=0$	bnd		F-B	$=0$	bnd	
23	F-B	$=0$	free	free	F-B	$=0$	free	free
24	C-M(f,x)	$=0$	none	free	C-M(f,x)	$=0$	free	free
25	C-M(f,x)	$=0$	none		C-M(f,x)	$=0$	bnd	
26	NCPF	$=0$	none	free	NCPF	$=0$	free	
27	NCPF	$=0$	none	free	Billups†	$=0$	none	free
28	NCPF	$=0$	bnd		NCPF	$=0$	bnd	
29	NCPF	$=0$	free	free	NCPF	$=0$	free	free
30	NCPCM(x,f)	$=0$	free	free	C-M(x,f)	$=0$	free	free
31	NCPCM(x,f)	$=0$	bnd		C-M(x,f)	$=0$	bnd	
32	NCPCM(f,x)	$=0$	none	free	NCPCM(f,x)	$=0$	free	free
33	NCPCM(f,x)	$=0$	none		NCPCM(f,x)	$=0$	bnd	

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: nlpec.gms) using one of the reformulation strategies from Section 3. 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 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 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 reformulation.

OQNLP and MSNLP

Optimal Methods Inc, 7134 Valburn Dr., Austin, TX 78731 www.optimalmethods.com, 512-346-7837

OptTek System, Inc., 1919 7th St., Boulder, CO 80302, www.opttek.com, 303-447-3255

Contents

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` keyword in Section 3, and Appendix B. 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 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 1default parameters and options (which specify 1000 iterations). It solved two of those three to within 1library to within 1seven 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

$$\text{minimize} \quad f(x,y) \tag{1.1}$$

subject to the nonlinear constraints

$$gl \leq G(x,y) \leq gu \tag{1.2}$$

and the linear constraints

$$l \leq A_1 x + A_2 y \leq u \tag{1.3}$$

$$x \in S, \quad y \in Y \tag{1.4}$$

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 1.1 - 1.4 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

$$P_1(x,w) = f(x) + \sum_{i=1}^{m} W_i viol(g_i(x)) \tag{1.5}$$

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 1.4. For simplicity, we assume there are no y variables: these would be fixed when this function is used. The function $viol(g_i(x))$ is equal to the absolute amount by which the ith constraint is violated at the point x. It is well known (see [Nash and Sofer, 1996]) that if x^* is a local optimum of 1.1 - 1.4, u^* is a corresponding optimal multiplier vector, the second order sufficiency conditions are satisfied at (x^*, u^*) , and

$$w_t > abs(u_i^*) \tag{1.6}$$

then x^* is a local unconstrained minimum of P_1 . If 1.1 - 1.4 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:

I Echos of important configuration and setup values

II Echo (optionally) of options file settings processed

III Echos of important algorithm settings, parameters, and termination criteria

IV The iteration log

V 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:

`Itn`	iteration number
`Penval`	Penalty function value
`Merit Filter`	ACC if the merit filter accepts the point, REJ if it rejects
`Merit Threshold`	threshold value for merit filter: accepts if Penval < Threshold
`Dist Filter`	ACC if the distance filter accepts the point, REJ if it rejects
`Best Obj`	Best feasible objective value found thus far
`Solver Obj`	Objective value found by NLP solver at this iteration
`Term Code`	Code indicating reason for termination of NLP solver: KTC means Kuhn-Tucker optimality conditions satisfied FRC means that the fractional objective change is less than a tolerance for some number of consecutive iterations INF means solver stopped at an infeasible point
`Sinf`	sum of infeasibilities at point found by NLP solver

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 th 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.

```
 Itn    Penval  Merit    Merit    Dist    Best      Solver    Term    Sinf
                Filter Threshold  Filter   Obj        Obj     Code
   0 +1.000e+030     -1.000e+030      -1.618e+002 -1.618e+002 FRC +0.000e+000
  20 -4.485e+000
  40 -6.321e+000
  60 -1.126e+001
  80 +2.454e+000
 100 +8.097e+001
 120 +5.587e+001
 140 +1.707e+004
 160 +2.034e+002
 180 +7.754e+001
 200 -6.224e+000

 Itn    Penval  Merit    Merit    Dist    Best      Solver    Term    Sinf
                Filter Threshold  Filter  Obj         Obj     Code
 201 +1.000e+030 ACC -1.000e+030  ACC -1.618e+002 -1.760e+002 FRC +0.000e+000
 207 -2.318e+001 ACC -2.075e+001  ACC -1.760e+002 -1.764e+002 FRC +0.000e+000
 220 -8.324e+000 REJ -2.318e+001  ACC -1.764e+002
 240 +8.351e+000 REJ -1.834e+001  ACC -1.764e+002
 251 -1.117e+001 ACC -1.008e+001  ACC -1.764e+002 -1.682e+002 FRC +0.000e+000
 256 -1.244e+001 ACC -1.117e+001  ACC -1.764e+002 -1.758e+002 FRC +0.000e+000
 258 -1.550e+001 ACC -1.244e+001  ACC -1.764e+002 -1.678e+002 FRC +0.000e+000
 260 -7.255e+000 REJ -1.550e+001  ACC -1.764e+002
 280 +8.170e+001 REJ -1.220e+001  ACC -1.764e+002
 282 -2.521e+001 ACC -1.220e+001  ACC -1.764e+002 -1.758e+002 FRC +0.000e+000
 300 +5.206e+001 REJ -2.521e+001  ACC -1.764e+002
 300 +5.206e+001 REJ -2.521e+001  ACC -1.764e+002
 320 +1.152e+000 REJ -1.642e+001  ACC -1.764e+002
 329 -2.111e+001 ACC -1.294e+001  ACC -1.764e+002 -1.763e+002 FRC +0.000e+000
 338 -3.749e+001 ACC -2.111e+001  ACC -1.764e+002 -1.763e+002 FRC +0.000e+000
 340 +2.235e+002 REJ -3.749e+001  ACC -1.764e+002
```

```
   360 +8.947e+001 REJ -2.363e+001  ACC -1.764e+002
   366 -3.742e+001 ACC -2.363e+001  ACC -1.764e+002 -1.761e+002 FRC +0.000e+000
   380 -2.244e+001 REJ -3.742e+001  ACC -1.764e+002
   391 -2.974e+001 ACC -2.244e+001  ACC -1.764e+002 -1.754e+002 FRC +0.000e+000
   400 +1.986e+002 REJ -2.974e+001  ACC -1.764e+002
   400 +1.986e+002 REJ -2.974e+001  ACC -1.764e+002
   420 -1.231e+001 REJ -2.359e+001  ACC -1.764e+002
   432 -2.365e+001 ACC -2.359e+001  ACC -1.764e+002 -1.766e+002 FRC +0.000e+000
   440 +6.335e+000 REJ -2.365e+001  ACC -1.766e+002
   460 -8.939e+000 REJ -1.872e+001  ACC -1.766e+002
   473 -3.216e+001 ACC -1.872e+001  ACC -1.766e+002 -1.775e+002 FRC +0.000e+000
   480 +1.744e+002 REJ -3.216e+001  ACC -1.775e+002
```

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:

`No.`	index of local solution
`Obj`	objective value of local solution
`Var`	variable index
`Value`	variable value

```
No. Obj           Var Value
1  -1.03163e+000  1  -8.98448e-002
1  -1.03163e+000  2   7.12656e-001
2  -1.03163e+000  1   8.98418e-002
2  -1.03163e+000  2  -7.12656e-001
3  -2.15464e-001  1   1.70361e+000
3  -2.15464e-001  2  -7.96084e-001
4  -2.15464e-001  1  -1.70361e+000
4  -2.15464e-001  2   7.96084e-001
5   0.00000e+000  1   0.00000e+000
5   0.00000e+000  2   0.00000e+000
6   2.10425e+000  1   1.60710e+000
6   2.10425e+000  2   5.68656e-001
7   2.10425e+000  1  -1.60711e+000
7   2.10425e+000  2  -5.68651e-001
```

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.

Option	Description	Default
ARTIFICIAL_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.	1.e4
BASIN_DECREASE_ FACTOR	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-BASIN_DECREASE_FACTOR) if WAITCYCLE consecutive trial points have distance from that solution less than MAXDIST.	0.2
BASIN_OVERLAP_FIX	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.	1
DISTANCE_FACTOR	If the distance between a trial point and any local solution found previously is less than DISTANCE_FACTOR $*$ 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.	1.0
DYNAMIC_DISTANCE_ FILTER	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 WAITCYCLE consecutive trial points have a their distances from that solution less than MAXDIST. MAXDIST is multiplied by (1-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.	1
DYNAMIC_MERIT_FILTER	A value of 1 turns on logic which dynamically varies the parameter which increases the merit filter threshold, THRESHOLD_INCREASE_FACTOR. If WAITCYCLE consecutive trial points have been rejected by the merit filter, this value is replaced by max(THRESHOLD_INCREASE_FACTOR, val), where val is the value of THRESHOLD_INCREASE_FACTOR which causes the merit filter to just accept the best of the previous 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.	1
ENABLE_SCREEN_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.	1
ENABLE_STATISTICS_ 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.	0

Option	Description	Default
FEASIBILITY TOLERANCE	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.	1.e-4
ITERATION_PRINT_ FREQUENCY	If the OQNLP iteration log is written to the GAMS log file, one line of output is written every k'th OptQuest iteration, where k is the value given here.	20
ITERATION_LIMIT	Increasing this limit can allow OQNLP to find a better solution. Try it if your run using 1000 iterations doesn't 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 an OQNLP run	1000
LOCALS_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.	No locals file created
LOCALS_FILE_FORMAT	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>`	REPORT
LOGFILE_ITN_PRINT_ FREQUENCY	In the output written to the log file, one line of output is written every k'th iteration, where k is the value given here.	20
MAX_LOCALS	When the number of distinct local solutions found exceeds the value specified here, the system will stop, returning the best solution found.	1000
MAX_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.	1000
MAX_SOLVER_CALLS_ NOIMPROVEMENT	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.	100
MAXTIME	When the execution time exceeds this value, the system will stop, returning the best solution found.	1000 seconds
NLPSOLVER	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.	LSGRG

Option	Description	Default
OPTQUEST_ONLY	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 aren't any).	0
OQNLP_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.	0
POINT_GENERATION	OPTQUEST causes trial points to be generated by the OptQuest driver RANDOM causes trial points to be generated by sampling each variable from a uniform distribution defined within its bounds SMARTRANDOM1 generates trial points by sampling each variable independently from either normal or triangular distributions, whose parameters are determined as described in Appendix A.	SMART-RANDOM1 (MSNLP) OPTQUEST (OQNLP)
SAMPLING_DISTRIBUTION	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.	0
SEARCH_TYPE	This option applies only to the OptQuest driver, and controls the search strategy used by OptQuest. The three choices that are relevant for use within OQNLP are: aggressive This choice controls the population update in step 7 of the OptQuest algorithm (see Appendix A). 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 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. The value of SEARCH_PARAMETER discussed below controls the fraction of points that are directed toward the boundary. crossover This option affects how OptQuest trial points are generated from population points. It retains the linear combination operator discussed in Appendix A, but adds a "crossover" operator, similar to those used in evolutionary or genetic algorithms, to create 2 additional trial points.	boundary
SOLVER_LOG_TO_GAMS_LOG	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 "???".	0
STAGE1_ITERATIONS	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.	200
THRESHOLD_INCREASE_FACTOR	This value must be nonnegative. If there are WAITCYCLE (see below) consecutive OptQuest iterations where the merit filter logic causes the NLP solver not to be called, the merit threshold is increased by mutipling it by (1+THRESHOLD_INCREASE_FACTOR)	0.2

Option	Description	Default
`USE_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.	`1`
`USE_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`).	`0`
`USE_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.	`1`
`WAITCYCLE`	This value must be a positive integer. If the merit filter is used, and there are `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` (see above). Increasing `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.	`20`

5 Use as a Callable System

MSNLP and OQNLP is also available as a callable system. It currently uses the LSGRG2 NLP solver as its local solver, but any other NLP solver could be included. 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.

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 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 UPDATE LOCALS($x_0, xf, w$)
FOR $i = 1, n1$ DO
  Call SP($xt(i)$)
  Evaluate P($xt(i), w$)
```

```
ENDDO
$xt^*$ = point yielding best value of P$(xt(i), w)$ over all stage one points, $(i = 1, 2, ..., n1)$.
call $L(xt^*, xf)$
Call UPDATE LOCALS$(xt^*, xf, w)$
threshold = P$(xt^*, w)$
STAGE 2
FOR $i = 1, n2$ DO
  Call SP$(xt(i))$
  Evaluate P$(xt(i), w)$
  Perform merit and distance filter tests:
  Call distance filter$(xt(i)$, dstatus)
  Call merit filter$(xt(i)$, threshold, mstatus)
  IF (dstatus and mstatus = "accept") THEN
    Call $L(xt(i), xf)$
    Call UPDATE LOCALS$(xt(i), xf, w)$
  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(xt) is called, and the L1 exact penalty value P(xt, w) is calculated. The user can set $n1$ 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, $n2$ 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 STAGE2_ITERATIONS sets $n2$.

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`**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 insure 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 `WAITCYCLE` consecutive iterations, the threshold is increased by the updating rule:

threshold $\leftarrow$ threshold +`THRESHOLD_INCREASE_FACTOR` *(1.0+abs(threshold))

where the default value of `THRESHOLD_INCREASE_FACTOR` is 0.2 and that for `WAITCYCLE` is 20. The additive 1.0 term is included so that threshold increases by at least `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 `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 `WAITCYCLE` or by increasing `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 `USE_DISTANCE_FILTER` and/or `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.

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(xt)

```
IF (first call) THEN
  Generate $k1$ (default 400) diverse points in $S$ and evaluate the exact penalty function P$(x, w)$ at each point.
  B=subset of $S$ with $k2$ (default 10) best P values
  FOR i = 1,nvars DO
    xmax(i) = max of component i of points in B
    xmin(i)= min of component i of points in B
    mu(i) = (xmax(i) + xmin(i))/2
    ratio(i) = (xmax(i) - xmin(i))/(1+buvar(i)-blvar(i))
    sigfactor = 2.0
    IF (ratio>0.7) sigfactor = f(ratio)
    sigma(i) = (xmax(i) - xmin(i))/sigfactor
  ENDDO
ENDIF
FOR i = 1,nvars DO
  Generate a normally distributed random variable rv(i) with mean mu(i) and standard
deviation sigma(i)
  If rv(i) is between blvar(i) and buvar(i), xt(i) = rv(i)
  If rv(i)<blvar(i), generate xt(i) uniformly between blvar(i) and xmin(i)
  If rv(i)>buvar(i), generate xt(i) uniformly between xmax(i) and buvar(i)
ENDDO
```

Return *xt*

This SRN generator attempts to find a subset, B, of *k*2 ”good” points, and generates most of its trial points *xt*, 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 [blvar(i), 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 [xmin(i), xmax(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 [xmin(i), xmax(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 *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 *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 (*ratio* ≤ 0.7), then the standard deviation 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 ratio should be small, say less than 0.1, so $xt(i)$ values near the bounds are very unlikely. If *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 textitratio > 0.999. Thus if *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 [xmin(i), xmax(i)]. When the interval [xmin(i), 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.

References

Floudas, C.A., et al. 1999. Handbook of Test Problems in Local and Global Optimization. Kluwer Academic Publishers.

Laguna, Manuel, R. Marti. 2003. Scatter Search, Methodology and Implementations in C. Kluwer Academic Publishers.

Lasdon, L., J. Plummer, Z. Ugray, and M. Bussieck. 2004. Improved Filters and Randomized Drivers for Multi-start Global Optimization, working paper, MSIS Department, McCombs College of Business, May 2004. Submitted to Mathematical Programming.

Nash, S. G., A. Sofer. 1996. Linear and Nonlinear Programming. McGraw-Hill Companies, Inc.

Smith, S., L. Lasdon. 1992. Solving Large Sparse Nonlinear Programs Using GRG. ORSA Journal on Computing 4 1 3-15.

Ugray, Z., L. Lasdon, J. Plummer, et.al. 2003. A Multistart Scatter Search Heuristic for Smooth NLP and MINLP problems, submitted to Informs Journal on Computing. Copies available on request from the author, or on the web at www.utexas.edu/courses/lasdon, link to papers.

OSL

Contents

1 Introduction

This document describes the GAMS interface to OSL.

OSL is the IBM Optimization Subroutine Library, containing high performance solvers for LP (linear programming), MIP (mixed integer programming) and QP (quadratic programming) problems. GAMS does not support the QP capabilities of OSL, you have to use a general non-linear solver like MINOS or CONOPT for that.

OSL offers quite a few algorithms and tuning parameters. Most of these are accessible by the GAMS user through an option file. In most cases GAMS/OSL should perform satisfactory without using any options.

2 How to Run a Model with OSL

OSL is capable of solve models of the following types: LP, RMIP and MIP. If you did not specify OSL as a default LP, RMIP or MIP solver, then you can use the following statement in your GAMS model:

`option lp = osl;` { or `RMIP` or `MIP` }

It should appear before the SOLVE statement.

3 Overview of OSL

OSL offers several algorithms for solving LP problems: a primal simplex method (the default), a dual simplex method, and three interior point methods: primal, primal-dual and primal-dual predictor-corrector. For network problems there is a network solver.

Normally the primal simplex method is a good method to start with. The simplex method is a very robust method, and in most cases you should get good performance with this solver. For large models that have to be solved many times it may be worthwhile to see if one of the other methods gives better results. Also changing the tuning parameters may influence the performance. The `method` option can be used to use another algorithm than the default one.

3.1 The Simplex Method

The most used method is the primal simplex method. It is very fast, and allows for restarts from an advanced basis. In case the GAMS model has multiple solves, and there are relatively minor changes between those LP models, then the solves after the first one will use basis information from the previous solve to do a 'jump start'. This is completely automated by GAMS and normally you should not have to worry about this.

In case of a 'cold start' (the first solve) you will see on the screen the message `'Crash...'`. This will try to create a better basis than the scratch ('all-slack') basis the Simplex method would normally get. The crash routine does not use much time, so it is often beneficial to crash. Crashing is usually not used for subsequent solves because that would destroy the advanced basis. The default rule is to crash when GAMS does not pass on a basis, and not to crash otherwise. Notice that in a GAMS model you can use the `bratio` option to influence GAMS whether or not to provide the solver with a basis. The default behavior can be changed by the `crash` option in the option file.

By default the model is also scaled. Scaling is most of the time beneficial. It can prevent the algorithm from breaking down when the matrix elements have a wide range: i.e. elements with a value of 1.0e-6 and also of 1.0e+6. It can also reduce the solution time. The presolver is called to try to reduce the size of the model. There are some simple reductions that can be applied before the model is solved, like replacing singleton constraints (i.e. `x =l= 5` ;) by bounds (if there was already a tighter bound on X we just can remove this equation). Although most modelers will already use the GAMS facilities to specify bounds (`x.lo` and `x.up`), in many cases there are still possibilities to do these reductions. In addition to these reductions OSL can also remove some redundant rows, and substitute out certain equations. The presolver has several options which can be set through the `presolve` option.

The presolve may destroy an advanced basis. Sometimes this will result in very expensive restarts. As a default, the presolve is not used if an advanced basis is available. If using the presolve procedure is more useful than the use of an advanced basis, one can still force a presolve by using an option file.

GAMS/OSL uses the order: scale, presolve, crash. There is no possibility to change this order unless you have access to the source of the GAMS/OSL program. After the model is solved we have to call the postsolver in case the presolver was used. The postsolver will reintroduce the variables and equations the presolve substituted out, and will calculate their values. This solution is an optimal solution, but not a basic solution. By default we call simplex again to find an optimal basis. This allows us to restart from this solution. It is possible to turn off this last step by the option `postsolve 0`.

Occasionally you may want to use the dual simplex method (for instance when the model is highly primal degenerate, and not dual degenerate, or when you have many more rows than columns). You can use the `method dsimplex` option to achieve this. In general the primal simplex (the default) is more appropriate: most models do not have the characteristics above, and the OSL primal simplex algorithm is numerically more stable.

The other options for the Simplex method like the refactorization frequency, the Devex option, the primal and the dual weight and the change weight option are only to be changed in exceptional cases.

3.2 The Interior Point Methods

OSL also provides you with three interior point solvers. You should try them out if your model is large and if it is not a restart. The primal-dual barrier method with predictor-corrector is in general the best algorithm. This can be set by `method interior3`. Note that the memory estimate of OSL is in many cases not sufficient to solve a model with this method. You can override OSL's estimate by adding to using the workspace model suffix as shown in Section 4.2. We have seen models where we had to ask for more than twice as much as the estimate. It is worthwhile to check how the interior points are doing on your model especially when your model is very large.

3.3 The Network Method

A linear model can be reformulated to a network model if

- The objective variable is a free variable.
- The objective variable only appears in the objective function, and not somewhere else in the model. This in fact defines the objective function.
- The objective function is of the form `=e=`.
- Each variable appears twice in the matrix (that is excluding the objective function) once with a coefficient of +1 and once with a coefficient of -1. In case there is a column with two entries that are the same, GAMS/OSL will try a row scaling. If there are no matrix entries left for a column (only in the objective function) or there is only one entry, GAMS/OSL will try to deal with this by adding extra rows to the model.

4 GAMS Options

The following GAMS options are used by GAMS/OSL.

4.1 Options Specified Through the Option Statement

The following options are specified through the option statement. For example,

```
set iterlim = 100 ;
```

sets the iterations limit to 100.

Option	**Description**
`iterlim`	Sets the iteration limit. 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.
`reslim`	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.
`optca`	Absolute optimality criterion for a MIP problem.
`optcr`	Relative optimality criterion for a MIP problem.
`bratio`	Determines whether or not to use an advanced basis.
`sysout`	Will echo the OSL messages to the GAMS listing file. This option is useful to find out exactly what OSL is doing.

4.2 Options Specified Through Model Suffixes

The following options are specified through the use of model suffix. For example,

```
mymodel.workspace = 10;
```

sets the amount of memory used to 10 MB. `Mymodel` 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.

Option	Description
`workspace`	Gives OSL x MB of workspace. Overrides the memory estimation.
`optfile`	Instructs OSL to read the option file *osl.opt*.
`cheat`	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 OSL option `improve` overrides the GAMS `cheat` parameter.
`cutoff`	Cutoff value. When the Branch and Bound search starts, the parts of the tree with an objective worse than x are deleted. Can speed up the initial phase in the Branch and Bound.
`prioropt`	Instructs OSL to use priority branching information passed by GAMS through the `variable.prior` parameters.

5 Summary of OSL Options

All these options should be entered in the option file *osl.opt* after setting the `mymodel.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.

5.1 LP Algorithmic Options

Option	Description
`crash`	crash an initial basis
`iterlim`	iteration limit
`method`	solution method
`pdgaptol`	barrier method primal-dual gap tolerance
`presolve`	perform presolve. Reduce size of model.
`reslim`	resource limit
`simopt`	simplex option
`scale`	perform scaling
`toldinf`	dual feasibility tolerance
`tolpinf`	primal feasibility tolerance
`workspace`	memory allocation

5.2 MIP Algorithmic Options

Option	Description
`bbpreproc`	branch and bound preprocessing
`cutoff`	cutoff or incumbent value
`cuts`	allocate space for extra cuts
`degscale`	scale factor for degradation
`improve`	required level of improvement over current best integer solution
`incore`	keep tree in core
`iweight`	weight for each integer infeasibility
`maxnodes`	maximum number of nodes allowed
`maxsols`	maximum number of integer solutions allowed
`optca`	absolute optimality criterion
`optcr`	relative optimality criterion

Option	Description
`strategy`	MIP strategy
`target`	target value for objective
`threshold`	supernode processing threshold
`tolint`	integer tolerance

5.3 Screen and Output File Options

Option	Description
`bastype`	format of basis file
`creatbas`	create basis file
`creatmps`	create MPS file
`iterlog`	iteration log
`mpstype`	type of MPS file
`networklog`	network log
`nodelog`	MIP log

5.4 Advanced LP Options

Option	Description
`adjac`	save storage on AA^T
`chabstol`	absolute pivot tolerance for Cholesky factorization
`chweight`	rate of change of multiplier in composite objective function
`chtinytol`	cut-off tolerance in Cholesky factorization
`crossover`	when to switch to simplex
`densecol`	dense column threshold
`densethr`	density threshold in Cholesky
`devex`	devex pricing method
`droprowct`	constraint dropping threshold
`dweight`	proportion of feasible objective used in dual infeasible solution
`factor`	refactorization frequency
`fastits`	switching frequency to devex
`fixvar1`	tolerance for fixing variables in barrier method when infeasible
`fixvar2`	tolerance for fixing variables in barrier method when feasible
`formntype`	formation of normal matrix
`maxprojns`	maximum number of null space projections
`muinit`	initial value for μ
`mulimit`	lower limit for μ
`mulinfac`	multiple of μ to add to the linear objective
`mufactor`	reduction factor for μ in primal barrier method
`nullcheck`	null space checking switch
`objweight`	weight given to true objective function in phase I of primal barrier
`pdstepmult`	step-length multiplier for primal-dual barrier
`pertdiag`	diagonal perturbation in Cholesky factorization
`possbasis`	potential basis flag
`postsolve`	basis solution required
`projtol`	projection error tolerance starting value for multiplier in composite objective function
`pweight`	
`rgfactor`	reduced gradient target reduction
`rglimit`	reduced gradient limit
`simopt`	simplex option
`stepmult`	step-length multiplier for primal barrier algorithm

5.5 Examples of GAMS/OSL Option File

The following option file *osl.opt* may be used to force OSL to perform branch and bound preprocessing, a maximum of 4 integer solutions and to provide a log of the branch and bound search at every node.

```
bbpreproc 1
maxsols 4
nodelog 1
```

6 Detailed Description of OSL Options

Option	Description	Default
`adjac`	Generation of AA^T `0:` Save storage on forming AA^T in the interior point methods. `1:` Use a fast way of computing AA^T.	`1`
`bastype`	Format of basis file `0:` do not write level values to the basis file `1:` write level values to the basis file	`0`
`bbpreproc`	Preprocess the Branch and Bound tree. `0:` Do not preprocess the 0-1 structure. `1:` Use super-nodes, copy matrix `2:` Regular branch-and-bound, copy matrix `3:` Use super-nodes, overwrite matrix `4:` Regular branch-and-bound, overwrite matrix The pre-processor examines only the 0-1 structure. On models with only general integer1 variables (i.e. integer variables with other bounds than 0 and 1) the preprocessor will not do any good. A message is written if this happens. The preprocessor needs space for extra cuts. If no space available, the branch-and-bound search may fail. Use the `cuts` option to specify how much extra room has to be allocated for additional cuts. Notice that the `presolve` already may reduce the size of the model, and so create extra space for additional cuts.	`0`
`chabstol`	Absolute pivot tolerance for the Cholesky factorization. Range - [1.0e-30, 1.0e-6]	`1.0e-15`
`chtinytol`	Cut-off tolerance in the Cholesky factorization. Range - [1.0e-30, 1.0e-6]	`1.0e-18`
`chweight`	Rate of change for multiplier in composite objective function. Range - [1e-12,1] This value determines the rate of change for `pweight` or `dweight`. It is a non-linear factor based on case-dependent heuristics. The default of 0.5 gives a reasonable change if progress towards feasibility is slow. A value of 1.0 would give a greater change, while 0.1 would give a smaller change, and 0.01 would give very slow change.	`0.5`
`Crash`	Crash an initial basis. This option should not be used with a network or interior point solver. The option is ignored if GAMS provides a basis. To tell GAMS never to provide the solver with a basis use '`option bratio = 1;`' in the GAMS program. `0:` No crash is performed `1:` Dual feasibility may not be maintained `2:` Dual feasibility is maintained `3:` The sum of infeasibilities is not allowed to increase `4:` Dual feasibility is maintained and the sum of infeasibilities is not allowed to increase The options to maintain dual feasibility do not have much impact due to the way GAMS sets up the model. Normally, only options 0 or 1 need be used.	`1`, if no basis provided by GAMS

Option	Description	Default
`creatbas`	Create a basis file in MPS style. String is the file name. Can only be used if GAMS passes on a basis (i.e. not the first solve, and if `BRATIO` test is passed). The basis is written just after reading in the problem and after the basis is setup. The option `bastype` determines whether values are written to the basis file. **On UNIX precede the file name by ../ due to a bug in OSL (see Section 7.3).**	None
`creatmps`	This option can be used to create an MPS file of the GAMS model. This can be useful to test out other solvers or to pass on problems to the developers. String is file name. Its should not be longer than 30 characters. Example: `creatmps trnsport.mps`. The option `mpstype` determines which type (1 or 2 nonzeroes per line) is being used. The MPS file is written just after reading in the problem, before scaling and presolve. On **UNIX precede the file name by ../ due to a bug in OSL (see Section 7.3)** . Examples of the MPS files generated by OSL are in the Appendix.	None
`crossover`	Switching to simplex from the Interior Point method. Use of this option can be expensive. It should be used if there is a need to restart from the optimal solution in the next `solve` statement. `0:` Interior point method does not switch to Simplex `1:` Interior point method switches to Simplex when numerical problems arise. `2:` Interior point method switches to Simplex at completion or when in trouble. `3:` As in 2, but also if after analyzing the matrix it seems the model is more appropriate for the Simplex method. `4:` Interior point immediately creates a basis and switches over to Simplex.	`1`
`cutoff`	Cutoff or incumbent value. Overrides the CUTOFF in GAMS.	None
`cuts`	Allocate space for extra cuts generated by Branch and Bound preprocessor.	`10 + m/10`, where `m` is the number of rows
`degscale`	The scale factor for all degradation. Range - [0.0, maxreal]	`1.0`
`densecol`	Dense column threshold. Columns with more non- zeroes than DENSECOL are treated differently. Range - [10, maxint]	`99,999`
`densethr`	Density threshold in Cholesky. If DENSETHR ≤ 0 then everything is considered dense, if DENSETHR ≥ 1 then everything is considered sparse. Range - [-maxreal, maxreal]	`0.1`
`devex`	Devex pricing `0:` Switch off Devex pricing (not recommended for normal use). `1:` Approximate Devex method. `2:` Primal: Exact Devex method, Dual: Steepest Edge using inaccurate initial norms. `3:` Exact Steepest Edge method. `-1,-2,-3:`As 1,2,3 for the dual. For the primal the default is used until feasible, and then mode 1,2 or 3. Devex pricing is only used by the simplex method. Devex pricing is used after the first "cheap" iterations which use a random pricing strategy. In the primal case a negative value tells OSL to use the non-default devex pricing strategy only in phase II (in phase I the default devex strategy is used then). For the primal 1 (the default) or -2 are usually good settings, for the dual 3 is often a good choice.	`1`
`Droprowct`	The constraint dropping threshold. Range - [1,30]	`1`
`dweight`	Proportion of the feasible objective that is used when the solution is dual infeasible. Range - [0.0,1.0]	`0.1`
`factor`	Refactorization frequency. A factorization of the basis matrix will occur at least each `factor` iterations. OSL may decide to factorize earlier based on some heuristics (loss of precision, space considerations) Range - [0,999]	`100+m/100`, where `m` is the number of rows

Option	Description	Default
`fastits`	When positive, OSL switches to Devex after `fastits` random iterations, (to be precise at the first refactorization after that) but before it will price out a random subset, with correct reduced costs. When negative, OSL takes a subset of the columns at each refactorization, and uses this as a working set. Range - [-maxint,+maxint]. By default OSL primal simplex initially does cheap iterations using a random pricing strategy, then switches to Devex either because the success rate of the random pricing becomes low, or because the storage requirements are becoming high. This option allows you to change this pricing change. This option is only used when simplex was used with `simopt 2` which is the default for models solved from scratch. For more information see the OSL manual.	`0`
`fixvar1`	Tolerance for fixing variables in the barrier method if the problem is still infeasible. Range - [0.0, 1.0e-3]	`1.0e-7`
`fixvar2`	Tolerance for fixing variables when the problem is feasible. Range - [0.0, 1.0e-3]	`1.0e-8`
`formntype`	Formation of normal matrix `0`: Do formation of normal matrix in the interior point methods in a way that exploits vectorization. Uses lots of memory, but will switch to option 2 if needed. `1`: Save memory in formation of normal matrix.	`0`
`Improve`	The next integer solution should be at least `improve` better than the current one. Overrides the `cheat` parameter in GAMS.	None
`Incore`	Keep tree in core `0`: The Branch & Bound routine will save tree information on the disk. This is needed for larger and more difficult models. `1`: The tree is kept in core. This is faster, but you may run out of memory. For larger models use `incore 0`, or use the `workspace` model suffix to request lots of memory. GAMS/OSL can not recover if you are running out of memory, i.e. we cannot save the tree to disk, and go on with `incore 0`.	`0`
`Iterlim`	Iteration limit. Overrides the GAMS `iterlim` option. Notice that the interior point codes require much less iterations than the simplex method. Most LP models can be solved with the interior point method with 50 iterations. MIP models may often need much more than 1000 LP iterations.	`1000`
`iterlog`	LP iteration log frequency. How many iterations are performed before a new line to the log file (normally the screen) is written. The log shows the iteration number, the primal infeasibility, and the objective function. The same log frequency is passed on to OSL, so in case you have `option sysout=on` in the GAMS source, you will see an OSL log in the listing file with the same granularity. The resource usage is checked only when an iteration log is written. In case you set this option parameter to a very large value, a resource limit overflow will not be detected.	`20`
`iweight`	Weight for each integer infeasibility. Range [0.0, maxreal]	`1.0`
`maxnodes`	Maximum number of nodes allowed	`9,999,999`
`maxprojns`	Maximum number of null space projections. Range - [1,10]	`3`
`maxsols`	Maximum number of integer solution allowed.	`9,999,999`

Option	Description	Default
`method`	Solution Method `psimplex:` Uses primal simplex method as LP solver `dsimplex:` Uses dual simplex `simplex:` Uses either the primal or dual simplex based on model characteristics `network:` Uses the network solver `interior0:` Uses an appropriate barrier method `interior1:` Uses the primal barrier method `interior2:` Uses the primal-dual barrier method `interior3:` Uses the primal-dual predictor-corrector barrier method. In case any of the simplex algorithms is used the listing file will list the algorithm chosen by OSL (this was not possible for `interior0`). the best interior point algorithm overall seems to be the primal-dual with predictor-corrector. Note that if an interior point method has been selected for a MIP problem, only the relaxed LP is solved by the interior point method.	`psimplex`
`Mpstype`	Format of MPS file `1:` one nonzero per line in the COLUMN section `2:` two nonzeroes per line in the COLUMN section.	`2`
`Mufactor`	The reduction factor for μ in the primal barrier method.Range - [1.0e-6,0.99999]	`0.1`
`muinit`	Initial value for μ in primal barrier method. Range - [1.0e-20, 1.0e6]	`0.1`
`mulimit`	Lower limit for μ. Range - [1.0e-6,1.0]	`1.0e-8`
`mulinfac`	Multiple of μ to add to the linear objective. Range - [0.0, 1.0]	`0.0`
`networklog`	Iteration log frequency for the network solver. See `iterlog`.	`100`
`nodelog`	MIP node log frequency. Node log is written to the screen when a new integer is found or after `nodelog` nodes. OSL writes a log line each node. This is captured in the status file, and displayed in the listing file when you have `option sysout=on`; in the GAMS source. The status file can become huge when many nodes are being examined.	`20`
`Nullcheck`	Null space checking switch. See the OSL manual.	None
`Objweight`	Weight given to true objective function in phase 1 of the primal barrier method. Range - [0.0, 1.0e8]	`0.1`
`optca`	Absolute optimality criterion. The definition of this criterion is the same as described in the GAMS manual.	`0.0`
`optcr`	Relative optimality criterion. The definition of this criterion is the same as described in the GAMS manual.	`0.10`
`pdgaptol`	Barrier method primal-dual gap tolerance. Range - [1.0e-12, 1.0e-1]	`1.0e-7`
`pdstepmult`	Step-length multiplier for primal-dual barrier. Range - [0.01, 0.99999]	`0.99995`
`pertdiag`	Diagonal perturbation in the Cholesky factorization. Range - [0.0, 1.0e-6]	`1.0e-12`
`possbasis`	Potential basis flag. If greater than zero, variables that are remote from their bounds are marked potentially basic. Range - [0,maxint]	`1`
`postsolve`	Basis solution required. If the presolver was not used this option is ignored. `0:` No basis solution is required. The reported solution will be optimal but it will not be a basis solution. This will require less work than being forced to find a basic optimal solution. `1:` Basis solution is required. After the `postsolve`, the Simplex method is called again to make sure that the final optimal solution is also a basic solution	`1`

Option	Description	Default
`presolve`	Perform model reduction before starting optimization procedure. This should not be with network solver. If by accident, all the constraints can be removed from the model, OSL will not be able to solve it and will abort. `presolve` can sometimes detect infeasibilities which can cause it to fail, in which case the normal solver algorithm is called for the full problem. `-1`: Do not use presolve `0`: Remove redundant rows, the variables summing to zero are fixed. If just one variable in a row is not fixed, the row is removed and appropriate bounds are imposed on that variable. `1`: As 0, and doubleton rows are eliminated (rows of the form $px_j + qx_k = b$). `2`: As 0, and rows of the form $x_1 = x_2 + x_3 ..., x > 0$ are eliminated. `3`: All of the above are performed. The listing file will report how successful the presolve was. The presolver writes information needed to restore the original to a disk file, which is located in the GAMS scratch directory. The postsolve routine will read this file after the smaller problem is solved, and then simplex is called again to calculate an optimal basis solution of the original model. If no basis solution is required use the option `postsolve 0`.	`0`: for cold starts, `-1`: for restarts
`Projtol`	Projection error tolerance. Range - [0.0, 1.0]	`1.0e-6`
`pweight`	Starting value of the multiplier in composite objective function. Range: [1e-12,1e10]. OSL uses in phase I when the model is still infeasible a composite objective function of the form *phase_I_objective + pweight * phase_I_objective* . The phase I objective has a +1 or -1 where the basic variables exceed their lower or upper bounds. This gives already a little bit weight to the phase II `objective.pweight` starts with 0.1 (or whatever you specify as starting value) and is decreased continuously. It is decreased by a large amount if the infeasibilities increase and by small amounts if progress to feasibility is slow.	`0.1`
`Reslim`	Resource limit. Overrides the GAMS reslim option.	`1000s`
`rgfactor`	Reduced gradient target reduction factor. Range - [1.0e-6,0.999999]	`0.1`
`rglimit`	Reduced gradient limit. Range - [0.0, 1.0]	`0.0`
`scale`	Scaling done on the model. Scaling cannot be used for network models. It is advised to use scaling when using the primal barrier method, the other barrier methods (primal-dual and primal- dual predictor-corrector) in general should not be used with scaling. `0`: Do not scale the model `1`: Scale the model	`1`, except for methods mentioned above
`simopt`	Simplex option `0`: Use an existing basis `1`: Devex pricing is used `2`: Random pricing is used for the first "fast" iterations, then a switch is made to Devex pricing `3`: Use previous values to reconstruct a basis. The dual simplex method can only have options 0 or 1.	`2`: if no basis provided by GAMS and crashing off, `0`: otherwise
`stepmult`	Step-length multiplier for primal barrier algorithm. Range - [0.01, 0.99999]	`0.99`

Option	Description	Default
`strategy`	MIP strategy for deciding branching. `1`: Perform probing only on satisfied 0-1 variables. This is the default setting in OSL. When a 0-1 variable is satisfied, OSL will do probing to determine what other variables can be fixed as a result. If this bit is not set, OSL will perform probing on all 0-1 variables. If they are still fractional, OSL will try setting them both ways and use probing to build an implication list for each direction. `2`: Use solution strategies that assume a valid integer solution has been found. OSL uses different strategies when looking for the first integer solution than when looking for a better one. If you already have a solution from a previous run and have set a cutoff value, this option will cause OSL to operate as though it already has an integer solution. This is beneficial for restarting and should reduce the time necessary to reach the optimal integer solution. `4`: Take the branch opposite the maximum pseudo-cost. Normally OSL will branch on the node whose smaller pseudo-cost is highest. This has the effect of choosing a node where both branches cause significant degradation in the objective function, probably allowing the tree to be pruned earlier. With this option, OSL will branch on the node whose larger pseudo-cost is highest. The branch taken will be in the opposite direction of this cost. This has the effect of forcing the most nearly integer values to integers earlier and may be useful when any integer solution is desired, even if not optimal. Here the tree could potentially grow much larger, but if the search is successful and any integer solution is adequate, then most of it will never have to be explored. `8`: Compute new pseudo-costs as variables are branched on. Pseudo-costs are normally left as is during the solution process. Setting this option will cause OSL to make new estimates, using heuristics, as each branch is selected. `16`: Compute new pseudo-costs for unsatisfied variables. Pseudo-costs are normally left as is during the solution process. Setting this option will cause OSL to make new estimates, using heuristics, for any unsatisfied variables' pseudo-costs in both directions. This is done only the first time the variable is found to be unsatisfied. In some cases, variables will be fixed to a bound by this process, leading to better performance in the branch and bound routine. This work is equivalent to making two branches for every variable investigated. `32`: Compute pseudo-costs for satisfied variables as well as unsatisfied variables. Here, if 16 is also on, OSL will compute new estimated pseudo-costs for the unsatisfied as well as the unsatisfied ones. Again, this is very expensive, but can improve performance on some problems. `strategy` can be a combination of the above options by adding up the values for the individual options. 48 for instance will select 16 and 32.	None
`Target`	Target value for the objective.	5% worse than the relaxed solution
`threshold`	Supernode processing threshold. Range [0.0, maxreal]	`0`
`tolpinf`	Primal infeasibility tolerance. Row and column levels less than this values outside their bounds are still considered feasible. Range: [1e-12,1e-1].	`1e-8`
`toldinf`	Dual infeasibility tolerance. Functions as optimality tolerance for primal simplex. Range: [1e-12,1e-1].	`1e-7`
`tolint`	Integer tolerance Range - [1.0e-12, 1.0e-1]	`1.0e-6`
`workspace`	Memory allocation. Overrides the memory estimation and the `workspace` model suffix. Workspace is defined in Megabytes.	None

7 Special Notes

This section covers some special topics of interest to users of OSL.

7.1 Cuts Generated During Branch and Bound

The OSL documentation does not give an estimate for the number of cuts that can be generated when the `bbpreproc` is used. By default we now allow #rows/10 extra rows to be added. Use the `cuts` option to change this. Also we were not able to find out how many cuts OSL really adds, so we can not report this. You will see a message on the screen and in the listing file when OSL runs out of space to add cuts. When this happens, I have seen the branch & bound fail later on with the message: OSL *data was overwritten*, so make sure your `cuts` option is large enough. For this reason we have as a default: no preprocessing.

Note that when `cuts` is 0 and `presolve` is turned on, there may still be enough room for the preprocessor to add cuts, because the presolve reduced the number of rows.

7.2 Presolve Procedure Removing All Constraints from the Model

The PRESOLVE can occasionally remove all constraints from a model. This is the case for instance for the first solve in [WESTMIP]. An artificial model with the same behavior is shown below. OSL will not recover from this. Turn off the presolve procedure by using the option `presolve -1` to prevent this from happening. An appropriate message is written when this happens.

```
Variable x ;
Equation e ;
e..  x =e= 1 ;
model m /all/ ;
option lp = osl ;
solve m using lp minimizing x ;
```

7.3 Deleting the Tree Files

The MIP solver EKKMSLV uses two files to store the tree. Due to a bug in OSL we are unable to store these files in the GAMS scratch directory (the open statement we issue is ignored by OSL). Therefore after you solved a MIP with OSL with INCORE 0 (the default), two files *fort.82* and *fort.83* would be in the current directory. The shell script *gamsosl.run* will try to overcome this by doing a **cd** to the scratch directory. If this fails, you will see an error message on the screen.

An undesirable side effect of this is that all options that relate to user specified files have to be preceded by ../ to have the file going to the directory where the user started GAMS from. If you do not do this, the file will go to the current directory, which is the scratch directory, and the file will be removed when GAMS terminates. The affected commands are `creatmps` and `creatmbas`. So if you want to write an MPS file with the name *myfile.mps*, use the option `creatmps ../myfile.mps`.

8 The GAMS/OSL Log File

The iteration log file that normally appears on the screen has the following appearance on the screen for LP problems:

```
Reading data...
 Starting OSL...
 Scale...
 Presolve...
```

```
    Crashing...
    Primal Simplex...
       Iter            Objective   Sum Infeasibilities
         20       2155310.000000          48470.762716
         40       1845110.000000          37910.387877
         60       1553010.000000          26711.895409
         80        191410.000000                   0.0
        100        280780.000000                   0.0
        120        294070.000000                   0.0
    Postsolve...
    Primal Simplex...
        121        294070.000000     Normal Completion
                                     Optimal
```

For MIP problems, similar information is provided for the relaxed problem, and in addition the branch and bound information is provided at regular intervals. The screen log has the following appearance:

```
   Reading data...
    Starting OSL...
    Scale...
    Presolve...
    Crashing...
    Primal Simplex...
       Iter            Objective   Sum Infeasibilities
         20            19.000000              8.500000
         40             9.500000              6.250000
    **** Not much progress: perturbing the problem
         60             3.588470              3.802830
         80             0.500000              2.000000
        100             2.662888              0.166163
        116             0.000000     Relaxed Objective
   Branch\& Bound...
       Iter  Nodes  Rel Gap       Abs Gap    Best Found
        276     19   n/a           6.0000        6.0000 New
        477     39   n/a           6.0000        6.0000
        700     59   n/a           6.0000        6.0000
        901     79   n/a           6.0000        6.0000
       1119     99  65.0000        5.9091        6.0000
       1309    119  65.0000        5.9091        6.0000
       1538    139  17.0000        5.6667        6.0000
       1701    159  17.0000        5.6667        6.0000
       1866    179  17.0000        5.6667        6.0000
       2034    199  17.0000        5.6667        6.0000
       2158    219  11.0000        5.5000        6.0000
       2362    239  11.0000        5.5000        6.0000
       2530    259   8.0000        5.3333        6.0000
       2661    275   5.0000        3.3333        4.0000 Done
   Postsolve...
    Fixing integer variables...
    Primal Simplex...
       2661             4.000000     Normal Completion
                                     Integer Solution
   The solution satisfies the termination tolerances
```

The branch and bound information consists of the number of iterations, the number of nodes, the current relative gap, the current absolute gap and the current best integer solution.

9 Examples of MPS Files Written by GAMS/OSL

This appendix shows the different output formats for MPS and basis files. We will not explain the MPS format or the format of the basis file: we will merely illustrate the function of the options `mpstype` and `bastype`. Running [TRNSPORT] with the following option file

```
mpsfile trnsport.mps
```

will result in the following MPS file:

```
NAME          GAMS/OSL
ROWS
 N  OBJECTRW
 E  R0000001
 L  R0000002
 L  R0000003
 G  R0000004
 G  R0000005
 G  R0000006
COLUMNS
    C0000001  R0000001     -0.225000   R0000002      1.000000
    C0000001  R0000004      1.000000
    C0000002  R0000001     -0.153000   R0000002      1.000000
    C0000002  R0000005      1.000000
    C0000003  R0000001     -0.162000   R0000002      1.000000
    C0000003  R0000006      1.000000
    C0000004  R0000001     -0.225000   R0000003      1.000000
    C0000004  R0000004      1.000000
    C0000005  R0000001     -0.162000   R0000003      1.000000
    C0000005  R0000005      1.000000
    C0000006  R0000001     -0.126000   R0000003      1.000000
    C0000006  R0000006      1.000000
    C0000007  OBJECTRW      1.000000   R0000001      1.000000
RHS
    RHS1      R0000002    350.000000   R0000003    600.000000
    RHS1      R0000004    325.000000   R0000005    300.000000
    RHS1      R0000006    275.000000
BOUNDS
 FR BOUND1    C0000007      0.000000
ENDATA
```

MPS names have to be 8 characters or less. GAMS names can be much longer, for instance: X("Seattle","New-York"). We don't try to make the names recognizable, but just give them labels like R0000001 etc. Setting the option `mpstype 1` gives:

```
NAME          GAMS/OSL
ROWS
 N  OBJECTRW
 E  R0000001
 L  R0000002
 L  R0000003
 G  R0000004
 G  R0000005
 G  R0000006
COLUMNS
    C0000001  R0000001     -0.225000
    C0000001  R0000002      1.000000
    C0000001  R0000004      1.000000
    C0000002  R0000001     -0.153000
    C0000002  R0000002      1.000000
    C0000002  R0000005      1.000000
    C0000003  R0000001     -0.162000
    C0000003  R0000002      1.000000
    C0000003  R0000006      1.000000
    C0000004  R0000001     -0.225000
    C0000004  R0000003      1.000000
    C0000004  R0000004      1.000000
    C0000005  R0000001     -0.162000
    C0000005  R0000003      1.000000
    C0000005  R0000005      1.000000
    C0000006  R0000001     -0.126000
    C0000006  R0000003      1.000000
    C0000006  R0000006      1.000000
    C0000007  OBJECTRW      1.000000
    C0000007  R0000001      1.000000
RHS
    RHS1      R0000002    350.000000
    RHS1      R0000003    600.000000
    RHS1      R0000004    325.000000
    RHS1      R0000005    300.000000
    RHS1      R0000006    275.000000
BOUNDS
 FR BOUND1    C0000007      0.000000
ENDATA
```

To illustrate the creation of a basis file, we first solve the transport model as usual, but we save work files so we can restart the job:

```
gams trnsport save=t
```

Then we create a new file called *t2.gms* with the following content:

```
transport.optfile=1;
solve trnsport using lp minimizing z;
```

and we run *gams t2 restart=t* after creating an option file containing the line `creatbas trnsport.bas`. This results in the following basis file being generated.

```
NAME
 XL C0000002  R0000001
 XU C0000004  R0000004
 XU C0000006  R0000005
 XU C0000007  R0000006
ENDATA
```

When we change the option to bastype 1 we get:

```
NAME
 BS R0000002                0.000000
 BS R0000003                0.000000
 LL C0000001                0.000000
 XL C0000002  R0000001      0.000000                 0.000000
 LL C0000003                0.000000
 XU C0000004  R0000004      0.000000                325.000000
 LL C0000005                0.000000
 XU C0000006  R0000005      0.000000                300.000000
 XU C0000007  R0000006      0.000000                275.000000
ENDATA
```

PATH 4.6

Michael C. Ferris

Todd S. Munson

Contents

1 Complementarity

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. This document is primarily concerned with the complementarity problem.

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 $2n$ inequalities a subset of n that will be satisfied as equations. These problems arise in a variety of disciplines including engineering and economics [20] 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

$$\begin{array}{rc} \min_{x \geq 0} & \sum_{(i,j)\in\mathcal{A}} c_{i,j} x_{i,j} \\ \text{subject to} & \sum_{j:(i,j)\in\mathcal{A}} x_{i,j} \leq s_i, \ \forall i \\ & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, \ \forall j. \end{array} \tag{1.1}$$

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

$$0 \leq p_i^s, \ s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}.$$

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:

$$0 \leq p_i^s \quad \perp \quad s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, \quad \forall i$$

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:

$$0 \leq p_j^d \quad \perp \quad \textstyle\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, \quad \forall j.$$

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,

$$0 \leq x_{i,j} \quad \perp \quad p_i^s + c_{i,j} \geq p_j^d, \quad \forall (i,j) \in \mathcal{A}.$$

We combine the three conditions into a single problem,

$$\begin{array}{rcll} 0 \leq p_i^s & \perp & s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, & \forall i \\ 0 \leq p_j^d & \perp & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, & \forall j \\ 0 \leq x_{i,j} & \perp & p_i^s + c_{i,j} \geq p_j^d, & \forall (i,j) \in \mathcal{A}. \end{array} \tag{1.2}$$

This model defines a linear complementarity problem that is easily recognized as the complementary slackness conditions [6] of the linear program (1.1). For linear programs the complementary slackness conditions are both necessary and sufficient for x to be an optimal solution of the problem (1.1). Furthermore, the conditions (1.2) are also the necessary and sufficient optimality conditions for a related problem in the variables (p^s, p^d)

$$\begin{array}{ll} \max_{p^s,p^d\geq 0} & \sum_j d_j p_j^d - \sum_i s_i p_i^s \\ \text{subject to} & c_{i,j} \geq p_j^d - p_i^s,\ \forall (i,j) \in \mathcal{A} \end{array}$$

termed the dual linear program (hence the nomenclature "dual variables").

Looking at (1.2) a bit more closely we can gain further insight into complementarity problems. A solution of (1.2) 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 (1.2) 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.

1.1.1 GAMS Code

The GAMS code for the complementarity version of the transportation problem is given in Figure 28.1; 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 (1.2). 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 (1.2) 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.lo(i) = 0;
```

```
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;
```

Figure 28.1: A simple MCP model in GAMS, `transmcp.gms`

or alternatively

```
    positive variable z;
```

Further information on the GAMS syntax can be found in [35]. *Note that GAMS requires the modeler to write `F(z) =g= 0` whenever the complementary variable is lower bounded, and does not allow the alternative form `0 =l= F(z)`.*

1.1.2 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 (1.2) to investigate other issues and facets of the problem at hand.

Demand in the model of Figure 28.1 is independent of the prices p. Since the prices p are variables in the complementarity problem (1.2), 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 28.1. For example, a linear demand function could be expressed using

$$\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j(1-p_j^d),\ \forall j.$$

Note that the demand is rather strange if p_j^d 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, 16].

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 (1.2) is replaced by

$$p_i^s(1+t_i) + c_{i,j} \geq p_j^d,\ \forall (i,j) \in \mathcal{A}.$$

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.

1.1.3 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:

(NCP) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ 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, $z_iF_i(z) = 0$. We frequently refer to this property as z_i is "complementary" to F_i. 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 $p \in \mathbf{R}^m$ and an activity level $y \in \mathbf{R}^n$ such that

$$\begin{array}{rclcr} 0 \leq y & \perp & L(p) & := & -A^Tp \geq 0 \\ 0 \leq p & \perp & S(p,y) & := & b + Ay - d(p) \geq 0 \end{array} \qquad (1.3)$$

```
$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;
```

Figure 28.2: Walrasian equilibrium as an NCP, `walras1.gms`

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.

1.2.1 GAMS Code

A GAMS implementation of (1.3) is given in Figure 28.2. 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].

1.2.2 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 (1.3). The syntax for carrying this out is shown in Figure 28.3 where we use the variables `d` to store the demand function referred to in the excess supply equation. The model `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.

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 28.3, 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 28.3. 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 28.3, the model statement could be replaced by

```
    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.

```
$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;
```

Figure 28.3: Walrasian equilibrium as an MCP, `walras2.gms`

1.2.3 Mixed Complementarity Problem

A mixed complementarity problem (MCP) is specified by three pieces of data, namely the lower bounds ℓ, the upper bounds u and the function F.

(MCP) Given lower bounds $\ell \in \{\mathbf{R} \cup \{-\infty\}\}^n$, upper bounds $u \in \{\mathbf{R} \cup \{\infty\}\}^n$ and a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that *precisely* one of the following holds for each $i \in \{1, \ldots, n\}$:

$$\begin{array}{lll} F_i(z) = 0 & \text{and} & \ell_i \leq z_i \leq u_i \\ F_i(z) > 0 & \text{and} & z_i = \ell_i \\ F_i(z) < 0 & \text{and} & z_i = u_i. \end{array}$$

These relationships define a general MCP (sometimes termed a rectangular variational inequality [25]). We will write these conditions compactly as

$$\ell \leq x \leq u \quad \perp \quad F(x).$$

Note that the nonlinear complementarity problem of Section 1.1.3 is a special case of the MCP. For example, to formulate an NCP in the GAMS/MCP format we set

```
     z.lo(I) = 0;
```

or declare

```
     positive variable z;
```

Another special case is a square system of nonlinear equations

(NE) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$ find $z \in \mathbf{R}^n$ such that

$$F(z) = 0.$$

In order to formulate this in the GAMS/MCP format we just declare

```
     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 28.3 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)..   -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 Section 1.3.1. 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 jth 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, 19, 20].

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 Section 1.1. 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:

I Add the following line to the `transmcp.gms` file prior to the `solve` statement

```
option mcp = path;
```

PATH will then be used instead of the default solver provided.

II 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 transmcp
```

Code	String	Meaning
1	Normal completion	Solver returned to GAMS without an error
2	Iteration interrupt	Solver used too many iterations
3	Resource interrupt	Solver took too much time
4	Terminated by solver	Solver encountered difficulty and was unable to continue
8	User interrupt	The user interrupted the solution process

Table 28.1: Solver Status Codes

Code	String	Meaning
1	Optimal	Solver found a solution of the problem
6	Intermediate infeasible	Solver failed to solve the problem

Table 28.2: Model Status Codes

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 Chapter 2 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.

1.3.1 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 28.4. We use "..." to indicate where we have omitted continuing similar output.

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 28.1 and Table 28.2 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.

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

```
               S O L V E      S U M M A R Y

     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

                      LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york    -0.225    -0.225     +INF     50.000
seattle  .chicago     -0.153    -0.153     +INF    300.000
seattle  .topeka      -0.162    -0.126     +INF       .

...

---- VAR X           shipment quantities in cases

                      LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york      .       50.000     +INF       .
seattle  .chicago       .      300.000     +INF       .

...

**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED
                             0  REDEFINED
                             0     ERRORS
```

Figure 28.4: Listing File for solving `transmcp.gms`

```
variables x;
equations d_f;

x.lo = 0;
x.up = 2;

d_f.. 2*(x - 1) =e= 0;

model first / d_f.x /;
solve first using mcp;
```

Figure 28.5: First order conditions as an MCP, `first.gms`

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 28.4 is a good case; when the model has infeasibilities, these can be found by searching for the string "INFES" as described above.

1.3.2 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 ℓ 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

$$0 \le x \le 2 \quad \perp \quad 2(x-1)$$

which has a unique solution, $x = 1$. Figure 28.5 provides correct GAMS code for this problem. However, if we accidentally write the valid equation

```
    d_f.. 0 =e= 2*(x - 1);
```

the problem given to the solver is

$$0 \leq x \leq 2 \quad \perp \quad -2(x-1)$$

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 28.3, 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

$$\begin{array}{rc} \min_x & c^T x \\ \text{subject to} & Ax = b, \; x \geq 0 \end{array}$$

we can write the first order optimality conditions as either

$$\begin{array}{lcl} 0 \leq x & \perp & -A^T\mu + c \\ \mu \text{ free} & \perp & Ax - b \end{array}$$

or, equivalently,

$$\begin{array}{lcl} 0 \leq x & \perp & -A^T\mu + c \\ \mu \text{ free} & \perp & b - Ax \end{array}$$

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, $\frac{1}{2}\|F(x)\|^2$ 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 [14] 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 1.1 and is using PATH 4.x to solve it. See Section 1.3 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 28.6 and Figure 28.7.

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 Section 2.4 below), then the following output is generated after the PATH banner.

```
Reading options file PATH.OPT
 > output_linear_model yes;
Options: Read: Line 2 invalid: hi_there;
Read of options file complete.
```

If the option reader encounters an invalid option (as above), it reports this but carries on executing the algorithm. Following this, the algorithm starts working on the problem.

2.1.1 Diagnostic Information

Some diagnostic information is initially generated by the solver at the starting point. Included is information about the initial point and function evaluation. The log file here tells the value of the largest element of the starting point and the variable where it occurs. Similarly, the maximum function value is displays along with the equation producing it. The maximum element in the gradient is also presented with the equation and variable where it is located.

The second block provides more information about the Jacobian at the starting point. These can be used to help scale the model. See Chapter 3 for complete details.

```
--- 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             1.0416e+03             0.0e+00 (demand.new-york)
    1     1     3     3 1.0029e+03  1.0e+00    1.0e+01 (demand.new-york)
pn_search terminated: no basis change.
```

Figure 28.6: Log File from PATH for solving `transmcp.gms`

```
Major Iteration Log
major minor  func  grad  residual    step  type prox    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)

...

   15     2    17    17 1.3972e-09  1.0e+00 SO 4.8e-06 1.3e-09 (demand.chicago)

FINAL STATISTICS
Inf-Norm of Complementarity . .  1.4607e-08 eqn: (rational.seattle.chicago)
Inf-Norm of Normal Map. . . . .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Minimum Map . . . .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Fischer Function. .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Grad Fischer Fcn. .  1.3247e-09 eqn: (rational.seattle.chicago)

FINAL POINT STATISTICS
Maximum of X. . . . . . . . . .  3.0000e+02 var: (x.seattle.chicago)
Maximum of F. . . . . . . . . .  5.0000e+01 eqn: (supply.san-diego)
Maximum of Grad F . . . . . . .  1.0000e+00 eqn: (demand.new-york)
                                            var: (x.seattle.new-york)

 ** EXIT - solution found.

Major Iterations. . . . 15
Minor Iterations. . . . 31
Restarts. . . . . . . . 0
Crash Iterations. . . . 1
Gradient Steps. . . . . 0
Function Evaluations. . 17
Gradient Evaluations. . 17
Total Time. . . . . . . 0.020000
Residual. . . . . . . . 1.397183e-09
--- Restarting execution
```

Figure 28.7: Log File from PATH for solving `transmcp.gms` (continued)

Code	Meaning
C	A cycle was detected.
E	An error occurred in the linear solve.
I	The minor iteration limit was reached.
N	The basis became singular.
R	An unbounded ray was encountered.
S	The linear subproblem was solved.
T	Failed to remain within tolerance after factorization was performed.

Table 28.3: Linear Solver Codes

2.1.2 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 exaple of the Crash Log can be seen in Figure 28.6. 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 Section 3.2.1 for further information. The column "step" corresponds to the steplength 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 2.4 gives mechanisms to affect the behavior of these steps.

2.1.3 Major Iteration Log

After the crash is completed, the main algorithm starts as indicated by the "Major Iteration Log" flag (see Figure 28.7). 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, 15], 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 28.3 explains the return codes for the linear solver and Table 28.4 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.

2.1.4 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.

```
Minor Iteration Log
minor      t           z     w     v   art  ckpts  enter    leave
    1  4.2538e-01      8     2     0     0     0 t[    0] z[   11]
    2  9.0823e-01      8     2     0     0     0 w[   11] w[   10]
```

Code	Meaning
B	A Backtracking search was performed from the current iterate to the Newton point in order to obtain sufficient decrease in the merit function.
D	The step was accepted because the Distance between the current iterate and the Newton point was small.
G	A gradient step was performed.
I	Initial information concerning the problem is displayed.
M	The step was accepted because the Merit function value is smaller than the nonmonotone reference value.
O	A step that satisfies both the distance and merit function tests.
R	A Restart was carried out.
W	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.

Table 28.4: Step Type Codes

```
    3  1.0000e+00     9     2     0     0     0 z[   10] t[    0]
```

`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 [17] 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.

2.1.5 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.

2.1.6 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 Chapter 3.

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 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 Table 28.5, Table 28.6, and Table 28.7. Note that only the first three characters of every word are significant.

GAMS controls the total number of pivots allowed via the `iterlim` option. If more pivots are needed for a particular model then either of the following lines should be added to the `transmcp.gms` file before the solve statement

```
option iterlim = 2000;
transport.iterlim = 2000;
```

Similarly if the solver runs out of memory, then the workspace allocated can be changed using

```
transport.workspace = 20;
```

The above example would allocate 20MB of workspace for solving the model.

Problems with a singular basis matrix can be overcome by using the `proximal_perturbation` option [3], and linearly dependent columns can be output with the `output_factorization_singularities` option. For more information on singularities, we refer the reader to Chapter 3.

As a special case, PATH can emulate Lemke's method [7, 31] for LCP with the following options:

Option	Default	Explanation
convergence_tolerance	1e-6	Stopping criterion
crash_iteration_limit	50	Maximum iterations allowed in crash
crash_merit_function	fischer	Merit function used in crash method
crash_method	pnewton	pnewton or none
crash_minimum_dimension	1	Minimum problem dimension to perform crash
crash_nbchange_limit	1	Number of changes to the basis allowed
crash_perturb	yes	Perturb the problem using pnewton crash
crash_searchtype	line	Searchtype to use in the crash method
cumulative_iteration_limit	10000	Maximum minor iterations allowed
gradient_searchtype	arc	Searchtype to use when a gradient step is taken
gradient_step_limit	5	Maximum number of gradient step allowed before restarting
interrupt_limit	5	Ctrl-C's required before killing job
lemke_rank_deficiency_iterations	10	Number of attempts made to fix rank-deficient basis during lemke start
lemke_start	automatic	Frequency of lemke starts (automatic, first, always)
major_iteration_limit	500	Maximum major iterations allowed
merit_function	fischer	Merit function to use (normal or fischer)
minor_iteration_limit	1000	Maximum minor iterations allowed in each major iteration
nms	yes	Allow line searching, watchdoging, and nonmonotone descent
nms_initial_reference_factor	20	Controls size of initial reference value
nms_maximum_watchdogs	5	Maximum number of watchdog steps allowed
nms_memory_size	10	Number of reference values kept
nms_mstep_frequency	10	Frequency at which m steps are performed

Table 28.5: PATH Options

Option	Default	Explanation
nms_searchtype	line	Search type to use (line, or arc)
output	yes	Turn output on or off. If output is turned off, selected parts can be turned back on.
output_crash_iterations	yes	Output information on crash iterations
output_crash_iterations_frequency	1	Frequency at which crash iteration log is printed
output_errors	yes	Output error messages
output_factorization_singularities	yes	Output linearly dependent columns determined by factorization
output_final_degeneracy_statistics	no	Print information regarding degeneracy at the solution
output_final_point	no	Output final point returned from PATH
output_final_point_statistics	yes	Output information about the point, function, and Jacobian at the final point
output_final_scaling_statistics	no	Display matrix norms on the Jacobian at the final point
output_final_statistics	yes	Output evaluation of available merit functions at the final point
output_final_summary	yes	Output summary information
output_initial_point	no	Output initial point given to PATH
output_initial_point_statistics	yes	Output information about the point, function, and Jacobian at the initial point
output_initial_scaling_statistics	yes	Display matrix norms on the Jacobian at the initial point
output_initial_statistics	no	Output evaluation of available merit functions at the initial point

Table 28.6: PATH Options (cont)

Option	Default	Explanation
output_linear_model	no	Output linear model each major iteration
output_major_iterations	yes	Output information on major iterations
output_major_iterations_frequency	1	Frequency at which major iteration log is printed
output_minor_iterations	yes	Output information on minor iterations
output_minor_iterations_frequency	500	Frequency at which minor iteration log is printed
output_model_statistics	yes	Turns on or off printing of all the statistics generated about the model
output_options	no	Output all options and their values
output_preprocess	yes	Output preprocessing information
output_restart_log	yes	Output options during restarts
output_warnings	no	Output warning messages
preprocess	yes	Attempt to preprocess the model
proximal_perturbation	0	Initial perturbation
restart_limit	3	Maximum number of restarts (0 - 3)
return_best_point	yes	Return the best point encountered or the absolute last iterate
time_limit	3600	Maximum number of seconds algorithm is allowed to run

Table 28.7: PATH Options (cont)

```
crash_method none;
crash_perturb no;
major_iteration_limit 1;
lemke_start first;
nms no;
```

If instead, PATH is to imitate the Successive Linear Complementarity method (SLCP, often called the Josephy Newton method) [28, 33, 32] for MCP with an Armijo style linesearch on the normal map residual, then the options to use are:

```
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;
```

Note that `nms_memory_size 1` and `nms_initial_reference_factor 1` turn off the nonmonotone linesearch, while `nms_mstep_frequency 1` turns off watchdoging [5]. `nms_searchtype line` forces PATH to search the line segment between the initial point and the solution to the linear model, while `merit_function normal` tell PATH to use the normal map for calculating the residual.

2.5 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 2.1 with additional output for preprocessing. The options are the same between the two versions.

2.6 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 [18].

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.

2.6.1 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

$$F(z) = 0, \ \ell \le z \le u,$$

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:

$$\begin{array}{rcl} \ell \le x \le u & \perp & -y \\ y \text{ free} & \perp & F(x). \end{array}$$

This is the MCP model passed on to the PATH solver.

3 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.

3.1 Formal Definition of MCP

The mixed complementarity problem is defined by a function, $F : D \to \mathbf{R}^n$ where $D \subseteq \mathbf{R}^n$ is the domain of F, and possibly infinite lower and upper bounds, ℓ and u. Let $C := \{x \in \mathbf{R}^n \mid \ell \le x \le u\}$, a Cartesian product of closed (possibly infinite) intervals. The problem is given as

$$MCP: \text{find } x \in C \cap D \text{ s.t. } F(x)^T y - x \geq 0,\ \forall y \in C.$$

This formulation is a special case of the variational inequality problem defined by F and a (nonempty, closed, convex) set C. Special choices of ℓ 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 $\ell \equiv 0$, $u \equiv +\infty$).

3.2 Algorithmic Features

We now describe some of the features of the PATH algorithm and the options affecting each.

3.2.1 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:

$$\left\| x - \pi(x), \left[\left(\frac{x_i - \ell_i}{\|\ell_i\| + 1} \right)_+ (F_i(x))_+ \right]_i, \left[\left(\frac{u_i - x_i}{\|u_i\| + 1} \right)_+ (-F_i(x))_+ \right]_i \right\|$$

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 $\Phi_i(x) := \phi(x_i, F_i(x))$ 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 Φ to the MCP format is given for example in [14].

In the context of nonlinear complementarity problems the generalized minimum map corresponds to the classic minimum map $\min(x, F(x))$. Furthermore, for NCPs the minimum map and the Fischer function are both local error bounds and were shown to be equivalent in [36]. Figure 28.10 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 Φ, namely $\Psi(x) := \frac{1}{2}\sum \phi(x_i, F_i)^2$, is continuously differentiable on $\mathbf{R}^n$ 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 2.4, the default merit function in PATH 4.x is the Fischer function. To change this behavior the `merit_function` option can be used.

3.2.2 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 `crash_method 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 `crash_perturb no`.

3.2.3 Nonmontone Searches

The first line of defense against convergence to stationary points is the use of a nonmonotone linesearch [23, 24, 15]. In this case we define a reference value, R^k and we use this value in test for sufficient decrease: test:

$$\Psi(x^k + t_k d^k) \leq \mathbf{R}^k + t_k \nabla\Psi(x^k)^T d^k.$$

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 κ is used to determine the initial set of acceptable merit function values. The value of κ defaults to 1 in the code and can be modified with the `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 $i_0 = \text{argmax } M_i$ 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^{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.

3.2.4 Linear Complementarity Problems

PATH solves a linear complementarity problem each major iteration. Let $M \in \Re^{n \times n}$, $q \in \Re^n$, and $B = [l, u]$ be given. $(\bar{z}, \bar{w}, \bar{v})$ solves the linear mixed complementarity problem defined by M, q, and B if and only if it satisfies the following constrained system of equations:

$$Mz - w + v + q = 0 \tag{3.4}$$
$$w^T(z - l) = 0 \tag{3.5}$$
$$v^T(u - z) = 0 \tag{3.6}$$
$$z \in B, w \in \Re^n_+, v \in \Re^n_+, \tag{3.7}$$

where $x + \infty = \infty$ for all $x \in \Re$ and $0 \cdot \infty = 0$ by convention. A triple, $(\hat{z}, \hat{w}, \hat{v})$, satisfying equations (3.4) - (3.6) 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:

$$Mz - w + v + Da + \frac{(1-t)}{s}(sr) + q = 0 \tag{3.8}$$
$$w^T(z - l) = 0 \tag{3.9}$$
$$v^T(u - z) = 0 \tag{3.10}$$
$$z \in B, w \in \Re^n_+, v \in \Re^n_+, a \equiv 0, t \in [0, 1] \tag{3.11}$$

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:

I $W = \{i \in \{1, \dots, n\} \mid \hat{z}_i = l_i \text{ and } \hat{w}_i > 0\}$

II $V = \{i \in \{1, \dots, n\} \mid \hat{z}_i = u_i \text{ and } \hat{v}_i > 0\}$

III $Z = \{1, \dots, n\} \setminus W \cup V$

Since $(\hat{z}, \hat{w}, \hat{v})$ is a complementary triple, $Z \cap W \cap V = \emptyset$ and $Z \cup W \cup V = \{1, \dots, n\}$. 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:

I t leaving at its upper bound.

II Any artificial variable.

III 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 that 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.

3.2.5 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.

3.3 Difficult Models

3.3.1 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 subsec-

```
positive variable x;
equations F;

F.. 1 / x =g= 0;

model simple / F.x /;

x.l = 1e-6;

solve simple using mcp;
```

Figure 28.8: GAMS Code for Ill-Defined Function

```
FINAL STATISTICS
Inf-Norm of Complementarity . .  1.0000e+00 eqn: (F)
Inf-Norm of Normal Map. . . . .  1.1181e+16 eqn: (F)
Inf-Norm of Minimum Map . . . .  8.9441e-17 eqn: (F)
Inf-Norm of Fischer Function. .  8.9441e-17 eqn: (F)
Inf-Norm of Grad Fischer Fcn. .  8.9441e-17 eqn: (F)

FINAL POINT STATISTICS
Maximum of X. . . . . . . . . .  8.9441e-17 var: (X)
Maximum of F. . . . . . . . . .  1.1181e+16 eqn: (F)
Maximum of Grad F . . . . . . .  1.2501e+32 eqn: (F)
                                            var: (X)
```

Figure 28.9: PATH Output for Ill-Defined Function

tions. We illustrate uses for the merit function information and final point statistics within the context of these problems.

3.3.1.1 Function Undefined We begin with a one-dimensional problem for which F is ill-defined at $x = 0$ as follows:

$$0 \le x \quad \perp \quad \tfrac{1}{x} \ge 0.$$

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 28.8. 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.

After setting the starting point, GAMS generates the model, and PATH proceeds to "solve" it. A portion of the output relating statistics about the solution is given in Figure 28.9. PATH uses the Fischer Function indicator as its termination criteria by default, but evaluates all of the merit functions given in Section 3.2.1 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.

To indicate the difference between the merit functions, Figure 28.10 plots them all for the simple example. We

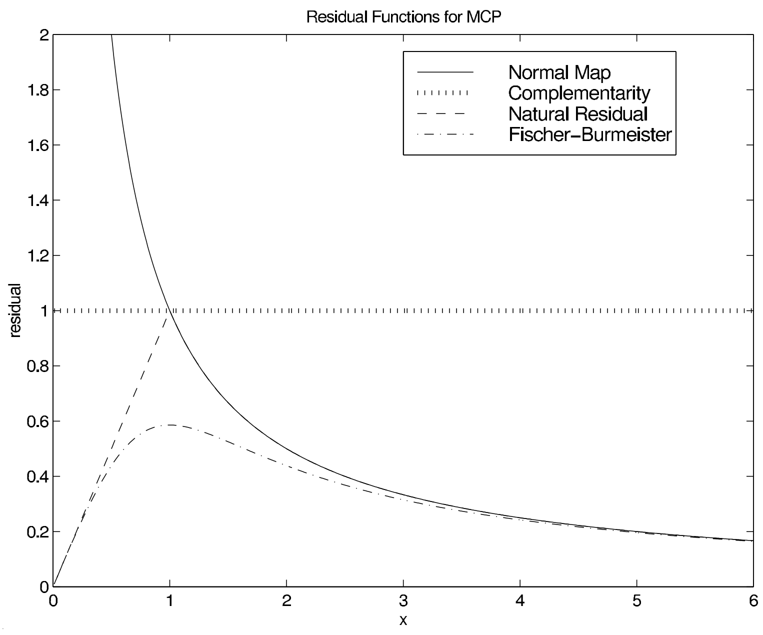


Figure 28.10: Merit Function Plot

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 ∇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

$$0 \leq x \quad \perp \quad \tfrac{1}{x+\epsilon} \geq 0$$

for a fixed $\epsilon > 0$, the function is well-defined over $C = \mathbf{R}^n_+$ 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 28.11 for $\epsilon = 1{*}10^{-6}$ and $x = 1{*}10^{-20}$. 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.

```
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 eqn: (G)
                                            var: (X)
```

Figure 28.11: PATH Output for Well-Defined Function

```
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 eqn: (F)
                                            var: (X)
```

Figure 28.12: PATH Output for Ill-Defined Jacobian

3.3.1.2 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 $x = 1 * 10^{-14}$, PATH generates the output given in Figure 28.12. We can see the that 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).

3.3.2 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

```
INITIAL POINT STATISTICS
Maximum of X. . . . . . . . . .  4.1279e+06 var: (w.29)
Maximum of F. . . . . . . . . .  2.2516e+00 eqn: (a1.33)
Maximum of Grad F . . . . . . .  6.7753e+06 eqn: (a1.29)
                                            var: (x1.29)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . . . .  9.4504e+06 eqn: (a2.29)
Minimum Row Norm. . . . . . . .  2.7680e-03 eqn: (g.10)
Maximum Column Norm . . . . . .  9.4504e+06 var: (x2.29)
Minimum Column Norm . . . . . .  1.3840e-03 var: (w.10)
```

Figure 28.13: PATH Output - Poorly Scaled Model

```
INITIAL POINT STATISTICS
Maximum of X. . . . . . . . . .  1.0750e+03 var: (x1.49)
Maximum of F. . . . . . . . . .  3.9829e-01 eqn: (g.10)
Maximum of Grad F . . . . . . .  6.7753e+03 eqn: (a1.29)
                                            var: (x1.29)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . . . .  9.4524e+03 eqn: (a2.29)
Minimum Row Norm. . . . . . . .  2.7680e+00 eqn: (g.10)
Maximum Column Norm . . . . . .  9.4904e+03 var: (x2.29)
Minimum Column Norm . . . . . .  1.3840e-01 var: (w.10)
```

Figure 28.14: PATH Output - Well-Scaled Model

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 28.13. The maximum row norm is defined as

$$\max_{1\leq i\leq n} \sum_{1\leq j\leq n} \mid (\nabla F(x))_{ij} \mid$$

and the minimum row norm is

$$\min_{1\leq i\leq n} \sum_{1\leq j\leq n} \mid (\nabla F(x))_{ij} \mid .$$

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 an 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:

```
    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 28.14 for the new model. 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.

```
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)
```

Figure 28.15: PATH Output - Zero Rows and Columns

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.

3.3.3 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 28.15. 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_singularities 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.

A 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.

A.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

$$u_a(d) = \prod_c d_c^{\alpha_{c,a}}, \quad \alpha_{c,a} \geq 0, \sum_c \alpha_{c,a} = 1,$$

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 of goods at those prices. The agents problem is therefore

$$\max_d u_a(d) \text{ subject to } p^T d \leq p^T e_a, d \geq 0,$$

where e_a is the endowment bundle for agent a. A closed form solution, corresponding to demand from agent a for commodity c is thus

$$d_{c,a}(p) := \frac{\alpha_{c,a} p^T e_a}{p_c}.$$

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

$$\min w_L y_L + w_r y_r \text{ subject to } \phi_c y_L^{\beta_c} y_r^{1-\beta_c} \geq x_{c,r},\ y_L, y_r \geq 0, \tag{A.12}$$

where ϕ_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

$$x_{c,r} \frac{\beta_c \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_L}.$$

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 (A.12) for unit demand $x_{c,r} \equiv 1$ and then multiplying these factor demands by the actual amount desired. Let $\bar{y}_L$ and $\bar{y}_r$ denote the optimal solutions of (A.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:

$$\begin{aligned}
\gamma_{c,r} &= w_L \bar{y}_L + w_r \bar{y}_r \\
&= w_L \frac{\beta_c \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_L} + w_r \frac{(1-\beta_c) \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_r} \\
&= \frac{1}{\phi_c} \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}.
\end{aligned}$$

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

$$T_{c,r}(w_p) := t_c d_r w_p,$$

where d_r is the distance of region r to the market. Spatial price equilibrium arises from the consideration:

$$0 \le x_{c,r} \perp \gamma_{c,r}(w_L, w_r) + T_{c,r}(w_p) \ge p_c.$$

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:

$$\gamma_{c,r} = \frac{1}{\phi_c}\left(\frac{w_L}{\beta_c}\right)^{\beta_c}\left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c} \tag{A.13}$$

$$0 \le x_{c,r} \quad \perp \quad \gamma_{c,r} + T_{c,r}(w_p) \ge p_c \tag{A.14}$$

$$0 \le w_L \quad \perp \quad e_L \ge \sum_{r,c} x_{c,r}\frac{\beta_c \gamma_{c,r}}{w_L} \tag{A.15}$$

$$0 \le w_r \quad \perp \quad a_r \ge \sum_{c} \frac{x_{c,r}(1-\beta_c)\gamma_{c,r}}{w_r} \tag{A.16}$$

$$0 \le w_p \quad \perp \quad e_P \ge \sum_{r,c} t_c d_r x_{c,r} \tag{A.17}$$

$$0 \le p_c \quad \perp \quad \sum_{r} x_{c,r} \ge \frac{\alpha_{c,P} e_P w_p + \alpha_{c,L} e_L w_L + \alpha_{c,O}\sum_r w_r a_r}{p_c} \tag{A.18}$$

Note that in (A.15), (A.16) and (A.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 (A.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 numeriare 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.

The first trial we attempted 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 3.3.1 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 (A.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 (A.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.

A.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

$$0 \leq i_r \quad \perp \quad w_r \geq 0.0001 * w_L$$

and modifies (A.16) and (A.18) as follows:

$$\begin{aligned} 0 \leq w_r \quad &\perp \quad a_r - i_r \geq \sum_c \frac{x_{c,r}(1-\beta_c)\gamma_{c,r}}{w_r} \\ 0 \leq p_c \quad &\perp \quad \sum_r x_{c,r} \geq \frac{\alpha_{c,P} e_P w_p + \alpha_{c,L} e_L w_L + \alpha_{c,O} \sum_r w_r(a_r - i_r)}{p_c}. \end{aligned} \tag{A.19}$$

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 (A.19), that can be avoided a posteori by imposing an artificial (inactive) lower bound on these prices. We chose not to do this however.

A.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. The intermediate good is "maintained land". Essentially, the following production problem replaces (A.12):

$$\begin{array}{rc} \min_{y_M, y_L, y_r, g} & w_L(y_M + y_L) + w_r y_r \\ \text{subject to} & y_r \geq (1 - \beta_c - \epsilon)g \\ & y_M \geq \epsilon g \\ & \phi_c y_L^{\beta_c} g^{1-\beta_c} \geq 1, \\ & y_M, y_L, y_r, g \geq 0. \end{array}$$

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 $\lambda_M = \frac{1}{\epsilon}$, ϵ 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 (A.14), (A.17), (A.18) and

$$\begin{aligned}
\gamma_{c,r} &= \frac{w_L^{\beta_c}}{\phi_c}\left(\frac{w_L\epsilon + w_r(1-\beta_c-\epsilon)}{1-\beta_c}\right)^{1-\beta_c} \\
0 \leq w_L \quad \perp \quad e_L &\geq \sum_{r,c} x_{c,r}\gamma_{c,r}\left(\frac{\beta_c}{w_L} + \frac{\epsilon(1-\beta_c)}{w_L\epsilon + w_r(1-\beta_c-\epsilon)}\right) \\
0 \leq w_r \quad \perp \quad a_r &\geq \sum_{c} \frac{x_{c,r}\gamma_{c,r}(1-\beta_c)(1-\beta_c-\epsilon)}{w_L\epsilon + w_r(1-\beta_c-\epsilon)}
\end{aligned}$$

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.

PATH References

[1] S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations.* PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, August 1995.

[2] S. C. Billups. Improving the robustness of descent-based mehtods for semi-smooth equations using proximal perturbations. *Mathematical Programming*, 87:153–176, 2000.

[3] S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. *Mathematical Programming*, 76:533–562, 1997.

[4] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide.* The Scientific Press, South San Francisco, CA, 1988.

[5] R. M. Chamberlain, M. J. D. Powell, and C. Lemaréchal. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Mathematical Programming Study*, 16:1–17, 1982.

[6] V. Chvátal. *Linear Programming.* W. H. Freeman and Company, New York, 1983.

[7] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications*, 1:103–125, 1968.

[8] R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem.* Academic Press, Boston, 1992.

[9] S. P. Dirkse. *Robust Solution of Mixed Complementarity Problems.* PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

[10] S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.

[11] S. P. Dirkse and M. C. Ferris. A pathsearch damped Newton method for computing general equilibria. *Annals of Operations Research*, pages 211–232, 1996.

[12] S. P. Dirkse and M. C. Ferris. Crash techniques for large-scale complementarity problems. In Ferris and Pang [19], pages 40–61.

[13] S. P. Dirkse and M. C. Ferris. Traffic modeling and variational inequalities using GAMS. In Ph. L. Toint, M. Labbe, K. Tanczos, and G. Laporte, editors, *Operations Research and Decision Aid Methodologies in Traffic and Transportation Management*, volume 166 of *NATO ASI Series F*, pages 136–163. Springer-Verlag, 1998.

[14] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming*, 86:475–497, 1999.

[15] M. C. Ferris and S. Lucidi. Nonmonotone stabilization methods for nonlinear equations. *Journal of Optimization Theory and Applications*, 81:53–71, 1994.

[16] M. C. Ferris, A. Meeraus, and T. F. Rutherford. Computing Wardropian equilibrium in a complementarity framework. *Optimization Methods and Software*, 10:669–685, 1999.

[17] M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12:207–227, 1999.

[18] M. C. Ferris and T. S. Munson. Preprocessing complementarity problems. Mathematical Programming Technical Report 99-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1999.

[19] M. C. Ferris and J. S. Pang, editors. *Complementarity and Variational Problems: State of the Art*, Philadelphia, Pennsylvania, 1997. SIAM Publications.

[20] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39:669–713, 1997.

[21] A. Fischer. A special Newton–type optimization method. *Optimization*, 24:269–284, 1992.

[22] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437–474, 1989.

[23] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23:707–716, 1986.

[24] L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, 59:779–805, 1991.

[25] P. T. Harker and J. S. Pang. Finite–dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications. *Mathematical Programming*, 48:161–220, 1990.

[26] G. W. Harrison, T. F. Rutherford, and D. Tarr. Quantifying the Uruguay round. *The Economic Journal*, 107:1405–1430, 1997.

[27] J. Huang and J. S. Pang. Option pricing and linear complementarity. *Journal of Computational Finance*, 2:31–60, 1998.

[28] N. H. Josephy. Newton's method for generalized equations. Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1979.

[29] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[30] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, Berkeley and Los Angeles, 1951.

[31] C. E. Lemke and J. T. Howson. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics*, 12:413–423, 1964.

[32] L. Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. *Mathematical Programming Study*, 23:144–162, 1985.

[33] L. Mathiesen. An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: An example. *Mathematical Programming*, 37:1–18, 1987.

[34] S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17:691–714, 1992.

[35] T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control*, 19:1299–1324, 1995.

[36] P. Tseng. Growth behavior of a class of merit functions for the nonlinear complementarity problem. *Journal of Optimization Theory and Applications*, 89:17–37, 1996.

[37] S. J. Wright. *Primal–Dual Interior–Point Methods.* SIAM, Philadelphia, Pennsylvania, 1997.

PATHNLP

Contents

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

```
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 Chapter 1, "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 Chapter 1, "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.

Option	Description	Default
`allow_reform`	Many models have an objective variable and equation that can be substituted out of the model, e.g. `z =E= f(x);` If this option is set, PATHNLP will substitute out the objective variable and equation where possible.	yes
`output_memory`	If set, the output will include a report on the memory allocated for use by PATHNLP.	`no`
`output_time`	If set, the output will include a report on the time used use by PATHNLP.	`no`
`skip_kkt`	If set, PATHNLP will skip the initial attempt to solve the KKT system for the NLP and go directly into a phase I / phase II method that first attempts to get feasible and then attempts to solve the KKT system starting from the feasible point found in phase I.	`no`

SBB

Contents

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 "Intermediate Nonoptimal". 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 `dfsstay` 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 `avgresmult` 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 `nodesel` option.
  - The `tryint` option now works for integer variables.
  - Additional information about node and variable selection can be printed to the Log file using the `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.

1 Introduction

SBB is a new 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	• Semicont	• Sos1
• Integer	• Semiint	• Sos2

2 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 subdivided, 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.

3 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.

4 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, reslim, 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 $x \geq 1$ (the node with $x \leq 0$ 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

```
   optname value
```

with one option on each line.

For example,

```
   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:

Option	Description	Default
`rootsolver`	`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.	GAMS NLP solver
`subsolver`	`solver[.n]` Similar to rootsolver but applied to the subnodes.	GAMS NLP solver
`loginterval`	The interval (number of nodes) for which log lines are written.	`1`
`loglevel`	The level of log output: `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	`1`
`subres`	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.	reslim
`subiter`	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.	iterlim
`failseq`	`solver1[.n1] solver2[.n2]` ... where `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, `solver2` is the name of a GAMS NLP solver that is used if `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 `solver.n` pairs. You can use the same solver several times with different options files. `failseq conopt conopt.2 conopt.3` means to try CONOPT with no options file. If this approach also fails, try CONOPT with options file *conopt.op2*, and if it again fails, try CONOPT with options file *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.	None
`infeasseq`	`level solver1[.n1] solver2[.n2]` ... 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 `<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.	None
`acceptnonopt`	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.	`0`

Option	Description	Default
avgresmult	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.	5
nodesel	Node selection: 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	0
dfsstay	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.	0
varsel	Variable selection: 0: automatic 1: maximum integer infeasibility 2: minimum integer infeasibility 3: pseudo costs	0
epint	The integer infeasibility tolerance.	1.0e-5
memnodes	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.	10000
printbbinfo	Additional info of log output: 0: no additional info 1: 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.	0
intsollim	maximum number of integer solutions. If this number is exceeded, SBB will terminate and return the best solution found so far.	99999

5 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.
   Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
      0     0   0    8457.6878   3           -        8457.6878       -
      1     1   1    8491.2869   2           -        8457.6878       -
      2     2   2    8518.1779   1           -        8457.6878       -
*     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:

Field	Description
`Node`	The number of the current node. The root node is node 0.
`Act`	The number of active nodes defined as the number of subnodes that have not yet been solved.
`Lev`	The level in the search tree, i.e., the number of branches needed to reach this node.
`Objective`	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).
`IInf`	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`.
`Best Int`	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.
`Best Bound`	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.
`Gap`	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:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12           -        6046.0186       -
   1     1   1   infeasible   -           -        6046.0186       -
   2     0   1    6042.0995  10           -        6042.0995       -
   3     1   2      ignored   -           -        6042.0995       - (4,6)
```

```
      4     0   2    5804.5856   8             -       5804.5856       -
      5     1   3      ignored   -             -       5804.5856       - (4,7)
```

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:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12            -       6046.0186       -
conopt.1 reports locally infeasible
Executing conopt1
conopt1 reports locally infeasible
Executing minos
minos reports locally infeasible
Executing snopt
   1     1   1  infeasible   -             -       6046.0186       -
   2     0   1   6042.0995  10             -       6042.0995       -
conopt.1 failed. 4 TERMINATED BY SOLVER, 7 INTERMEDIATE NONOPTIMAL
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
```

6 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).

SCENRED

Contents

Release Notes

- May, 2002: Level 001 (GAMS Distribution 20.6)
  - GAMS/SCENRED introduced.

1 Introduction

Stochastic programs with recourse employing a discrete distribution of the random parameters become a deterministic programming problem. They can be solved by an appropriate optimization algorithm, ignoring the stochastic nature of (some or all) parameters.

SCENRED is a tool for the reduction of scenarios modeling the random data processes. The scenario reduction algorithms provided by SCENRED determine a scenario subset (of prescribed cardinality or accuracy) and assign optimal probabilities to the preserved scenarios. The reduced problem is then solved by a deterministic optimization algorithm provided by GAMS.

2 Scenario Reduction Algorithms

Many solution methods for stochastic programs employ discrete approximations of the uncertain data processes by a set of scenarios (i.e., possible outcomes of the uncertain parameters) with corresponding probabilities.

For most practical problems the optimization problem that contains all possible scenarios (the so-called deterministic equivalent program) is too large. Due to computational complexity and to time limitations this program is often approximated by a model involving a (much) smaller number of scenarios.

The reduction algorithms developed in [1,2] determine a subset of the initial scenario set and assign new probabilities to the preserved scenarios. All deleted scenarios have probability zero.

SCENRED contains three reduction algorithms: The Fast Backward method, a mix of Fast Backward/Forward methods and a mix of Fast Backward/Backward methods. In general, the computational performance (accuracy, running time) of the methods differ. For huge scenario trees the Fast Backward method has the best expected performance with respect to running time. The results of the Forward and Backward methods are more accurate, but at the expense of higher computing time. The Forward method is the best algorithm when comparing accuracy, but it can only be recommended if the number of preserved scenarios is small (strong reduction). The combined methods improve the result of the Fast Backward method if the Forward or Backward method, respectively, can be completed within the running time limit. If no reduction method is selected, the method with the best expected performance with respect to running time is chosen.

The reduction algorithms exploit a certain probability distance of the original and the reduced probability measure. The probability distance trades off scenario probabilities and distances of scenario values. Therefore, deletion will occur if scenarios are close or have small probabilities.

The reduction concept is general and universal. No requirements on the stochastic data processes (e.g. the dependency or correlation structure of the scenarios, the scenario probabilities or the dimension of the process) or on the structure of the scenarios (e.g. tree-structured or not) are imposed. The reduction algorithms can be tailored to the stochastic model if the user provides additional information (How many decision stages are involved? Where do the random parameters enter the model – in objective and/or right hand sides and/or technology matrices?) The information is used to choose the probability distances (cf. Remark 1 in [1]).

References (download: `www-iam.mathematik.hu-berlin.de/~romisch/RecPubl.html`)

I J. Dupačová, N. Gröwe-Kuska, W. Römisch: Scenario reduction in stochastic programming: An approach using probability metrics. Revised version to appear in Mathematical Programming.

II H. Heitsch, W. Römisch: Scenario reduction algorithms in stochastic programming. Preprint 01-8, Institut für Mathematik, Humboldt-Universität zu Berlin, 2001.

3 Using GAMS/SCENRED

The reduction algorithms require additional data preparation and reformulation of the GAMS program for the stochastic programming model.

GAMS offers great flexibility with respect to the organization of data specification, model definition and solve statements. The most common way to organize GAMS/SCENRED programs is shown below. Since the initial scenarios and a number of input parameters have to be passed to SCENRED, the corresponding components of the GAMS program have to be defined before the SCENRED call. The reduced scenarios have to be defined before the equations of the (reduced) stochastic programming model are used in a solve statement. Therefore the SCENRED call can be placed anywhere between the definitions of the GAMS parameters and the solve statement of the reduced stochastic programming model.

When building or modifying a model for use with GAMS/SCENRED the following steps should be taken:

- Analyse the GAMS program of the stochastic programming model.
  Since the initial scenarios and a number of input parameters have to be passed to SCENRED (see Section 4), one must identify the corresponding components of the GAMS model and create or calculate them if they do not already exist.

- Reformulate the GAMS program.
  Check if the model can handle varying scenario or node probabilities, and whether the equations are defined in terms of a (possibly reduced) tree. If the model doesn't already contain a scenario tree, one should be added. If it does, it is a simple task to rewrite the equation definitions (and possibly other statements too) in terms of a subset of the original nodes or tree.

- Add the statements for passing the initial set of scenarios to SCENRED, for the execution of SCENRED and for the import of the reduced scenarios from SCENRED.

A reduction of the initial scenarios makes sense only if we are able to generate that part of the model that corresponds to the preserved scenarios (i.e. the reduced subtree). This is done by declaring a subset of the nodes in the original tree. The parameters and equations are declared over the original node set, but are defined over only the subtree. This will be illustrated by an example later in the section.

Further, one should verify that the model can handle changing probabilities. Many practical models involve scenarios with equal probabilities. This property will not be maintained by the probabilities in the reduced subtree.

ORGANIZATION OF GAMS/SCENRED PROGRAMS

Component	Contents
1. DATA	∘ set & parameter declarations and definitions ∘ `$libinclude scenred.gms` ∘ assignments ∘ displays
2. SCENRED CALL	∘ export the initial scenarios from GAMS to SCENRED ∘ execute SCENRED ∘ import the reduced scenarios from SCENRED to GAMS
3. MODEL	∘ variable declaration ∘ equation declarations ∘ equation definitions (using sets from reduced tree) ∘ model definition & solution

Prior to calling SCENRED, you should include the declaration of the SCENRED input and output parameters and the definition of the sets they are indexed by from the GAMS include library:

```
    $libinclude scenred.gms
```

Once you have created all the inputs to SCENRED and assigned values to them, you are ready to write the SCENRED GDX data input file, write the SCENRED options file, call SCENRED, and read the reduced tree data from the SCENRED GDX data output file (see Sections 4,5,6). Assuming your model is formulated to use a subtree of the original, you can now continue with the solve and any subsequent reporting.

SCENRED is executed by issuing the statement

```
   execute 'scenred optfilename';
```

where `optfilename` is the name of the SCENRED option file.

As an example, consider the `srkandw` model in the GAMS model library, and the `kand` model upon which it is based (get these from the modlib now!). To produce `srkandw` from `kand`, we first reformulate the original to allow for solution over a reduced tree. To do this, we introduce a subset of the node set: `set sn(n) 'nodes in reduced tree';` For convenience and clarity, we introduce a second subset at the same time, the set of leaf nodes: `set leaf(n) 'leaf nodes in original tree';` as well as some code to compute this set based on the existing time-node mapping. We also declare a new parameter, the probabilities for the reduced tree: `parameter sprob(n) 'node probability in reduced tree';` Once these are declared, we can quickly edit the equation *definitions* so that they run only over the reduced subtree: we simply substitute the reduced probabilities `sprob` for the original `prob`, and the reduced node set `sn` for the original node set `n`. Note that the *declaration* of the equations does not change.

This example illustrates one other change that may be required: the stochastic data must be in parameters having the node set as their last index. This is not the case in the `kand` model, so we simply reversed the indices in the `dem` parameter to meet the requirement in `srkandw`. It is also possible to create a transposed copy of the original data and pass that the SCENRED if the original data cannot be changed conveniently.

4 The SCENRED Input File

The SCENRED input file contains the initial scenarios and their stochastic parameter data, as well as statistics describing this input and (possibly) options to control the SCENRED run. This input file has a special binary format; it is a GDX (GAMS Data Exchange) file. The name of the SCENRED input file is assigned in the option file (see Section 5).

The scalar inputs to SCENRED are collected in the one-dimensional parameter `ScenRedParms`, the first parameter stored in the SCENRED input file. Some of the elements of `ScenRedParms` are required (e.g. statistics for the input tree) while others are optional (e.g. the run time limit). SCENRED will stop if a required element is missing or out of range.

Element	**Description**
`num_leaves`	the number of initial scenarios or leaves of the scenario tree (i.e., before the reduction)
`num_nodes`	number of nodes in the initial tree (the number of scenarios if not tree-structured)
`num_random`	Number of random variables assigned to a scenario or node, i.e., the dimension of the random data process
`num_time_steps`	Length of a path from the root node to a leaf of the scenario tree, i.e., the number of time steps involved

Table 31.1: Required `ScenRedParms` elements

Element	**Description**	**Default**
`red_num_leaves`	specifies the desired number of preserved scenarios or leaves	none
`red_percentage`	specifies the desired reduction in terms of the relative distance between the initial and reduced scenario trees (a real between 0.0 and 1.0)	none
`num_stages`	Set the number of branching levels of the scenario tree, i.e., the number of stages of the model -1. Hence `num_stages=1` if no branching occurs, i.e., the values of the scenarios differ for all time steps.	1
`where_random`	An integer indicating where the randomness enters the model. The value is interpreted as a “digit map” computed using the formula *100*inObj + 10*inRHS + inMatrix*, where *inObj* is 1 if the objective contains random parameters and 0 otherwise, *inRHS* is 1 if the right-hand side contains random parameters and 0 otherwise, and *inMatrix* is 1 if the constraint matrix contains random coefficients and 0 otherwise.	`10` (random right-hand side)
`reduction_method`	Select a reduction method: `0:` automatic (best expected performance with respect to running time) `1:` Fast Backward method `2:` Mix of Fast Backward/Forward methods `3:` Mix of Fast Backward/Backward methods	`0`
`run_time_limit`	Defines a limit on the running time in seconds	none
`report_level`	Control the content of the SCENRED log file: `0:` Standard SCENRED log file `1:` Additional information about the tree	`0`

Table 31.2: Optional `ScenRedParms` elements

A few comments on the parameters `red_percentage` and `red_num_leaves` are in order. At least one of these values must be set. The value of `red_percentage` will be ignored if the parameter `red_num_leaves` is non-zero. Otherwise, the tree will not be reduced if `red_percentage=0`, while the reduction of the tree will be maximal (i.e. only one scenario will be kept) if `red_percentage=1`. A numeric value of 0.5 means that the reduced tree maintains 50% of the information contained in the original tree. The reduction algorithms are skipped if `red_num_leaves=num_leaves` or if `red_num_leaves=0` and `red_percentage=0`. These values can be assigned if the user wishes to run the scenario tree diagnostic.

The second data element in the input file is the set of nodes making up the scenario tree. Note that the cardinality of this set is part of `ScenRedParms`.

The third data element is the ancestor mapping between the nodes. This mapping determines the scenario tree.

Note that the mapping can be either an ancestor mapping (i.e. child-parent) or a successor mapping (parent-child). By default, SCENRED expects an ancestor mapping. If the check for this fails, it looks for a successor mapping.

The fourth data element is the parameter of probabilities for the nodes in the original tree. It is only required that probabilities for the scenarios (i.e. the leaf nodes) be provided, but the parameter can contain probabilities for the non-leaf nodes as well.

The remaining elements in the input data file specify the parameter(s) that comprise the random values assigned to the initial scenarios, or to the nodes of the scenario tree. There can be more than one such parameter, included in any order. The only requirement is that the node set be the final index in each of these parameters.

Table 31.3 summarizes the content of the SCENRED input file. Please keep in mind that the order of the entries must not be altered!

No.	Symbol	Type	Dimension	Content
1	`ScenRedParms`	Parameter	1	scalar SCENRED input
2	(any name)	Set	1	nodes in the scenario tree
3	(any name)	Set	2	the ancestor set
4	(any name)	Parameter	1	node probabilities; at least for the leaves
≥5	(any name)	Parameter	≥ 1	random values assigned to the nodes

Table 31.3: Content of the SCENRED Input File

To create the SCENRED data input file, the GAMS **execute_unload** statement is used. This statement is used to transfer GAMS data to a GDX file at execution time. As an example, to create a GDX file with the 4 required input parameters and one parameter **demand** containing the stochastic data, you might have the following statement:

```
execute_unload 'sr_input.gdx', ScenRedParms, node, ancestor, prob, demand
```

5 SCENRED Options and the Option File

When the SCENRED executable is run, it takes only one argument on the command line: the name of the SCENRED option file. The option file is a plain text file. Typically, it is used to specify at least the names of the SCENRED data input and output files. The option file must be created by the SCENRED user (typically via the GAMS put facility during the GAMS run). The syntax for the SCENRED option file is

`optname value` or `optname = value`

with one option on each line. Comment lines start with an asterix and are ignored.

Some of the SCENRED options may be specified in two places: as elements of the **ScenRedParms** parameter of the SCENRED input file, or as entries in the options file. These parameters have been summarized in Table 31.2. If an option is set in both these places, the value in the option file takes precedence of over the value from **ScenRedParms**. In addition, the parameters in Table 31.4 can only be specified in the option file.

Option	Description	Default
`input_gdx`	Name of the SCENRED data input file	`xllink.gdx`
`output_gdx`	Name of the SCENRED data output file	`scenred.gdx`
`log_file`	Name of the SCENRED log file	`scenred.log`

Table 31.4: Options - optfile only

6 The SCENRED Output File

The SCENRED output file contains the reduced scenario tree and the `ScenRedReport` parameter. Like the input file, the output file has a special binary format; it is a GDX (GAMS Data Exchange) file.

The first data element in the output file is the `ScenRedReport` parameter containing the scalar outputs and statistics from the SCENRED run. The elements of this parameter are summarized in Table 31.5. The second data element is the parameter containing the probabilities of the nodes in the reduced scenario tree. These node probabilities are required to construct the reduced tree. The third and final data element is the ancestor map for the reduced scenario tree. This map can be read from the GDX file, or the reduced tree can be built from the original one by using the reduced probablilities. The content of the data output file is summarized in Table 31.6.

Element	Description
`ScenRedWarnings`	number of SCENRED warnings
`ScenRedErrors`	number of SCENRED errors
`run_time`	running time of SCENRED in sec.
`orig_nodes`	number of nodes in the initial scenario tree
`orig_leaves`	number of leaves (scenarios) in the initial scenario tree
`red_nodes`	number of nodes in the reduced scenario tree
`red_leaves`	number of leaves(scenarios) in the reduced tree
`red_percentage`	relative distance of initial and reduced scenario tree
`red_absolute`	absolute distance between initial and reduced scenario tree
`reduction_method`	reduction method used: `0:` the program stopped before it could select a method `1:` Fast Backward method `2:` Mix of Fast Backward/Forward methods `3:` Mix of Fast Backward/Backward methods

Table 31.5: `ScenRedReport` elements

No.	Symbol	Type	Dimension	Content
1	`ScenRedReport`	Parameter	1	report of the SCENRED run
2	`red_prob`	Parameter	1	node probabilities for the reduced scenarios
3	`red_ancestor`	Set	2	the ancestor map for the reduced scenarios

Table 31.6: Content of the SCENRED Output File

To read the SCENRED data output file, the GAMS `execute_load` statement is used. This statement is used to transfer GDX data to GAMS at execution time. As an example, to read a GDX file named `sr_output.gdx` created by SCENRED, you might have the following statement:

```
execute_load 'sr_output.gdx', ScenRedReport, sprob=red_prob, sanc=red_ancestor
```

In the statement above, the equal sign "=" is used to indicate that the data in the GDX parameter `red_prob` should be read into the GAMS parameter `sprob`, and the data in the GDX set `red_ancestor` should be read into the GAMS set `sanc`.

7 Diagnostic Check of Scenario Trees

When SCENRED reads its input data, it performs a number of checks to verify that the data is correct. The diagnostic checks of the input parameters include:

- consistency of the desired input parameters with the contents of the SCENRED input file (number of nodes, number of leaves, number of time steps, number of random values assigned to a node)

- range check of desired input parameters and options
- check of scenario and node probabilities
- check of the ancestor matrix (check the orientation of the graph, check if the graph contains a cycle, check if the graph contains incomplete forests or scenarios, check the consistency of the parameter `num_time_steps` with the ancestor matrix)

The following errors in the specification of the scenario tree cause SCENRED to skip the reduction algorithms:

- The input files cannot be opened.
- Not all required input parameters are given.
- The required input parameters are not consistent with the contents of the SCENRED input file.
- The required input parameters are out of range.
- Missing or negative scenario probabilities (probabilities of leaves).
- The ancestor set contains too many entries (more than `2*num_nodes`).
- SCENRED detects a cycle in the ancestor set.
- SCENRED detects incomplete scenarios in the ancestor set.
- Run time limit reached

8 SCENRED Errors and Error Numbers

When SCENRED encounters a serious error in the input files or in the scenario tree, it sends an error message to the screen and to the log file. These messages always start with

```
**** SCENRED run-time error ...
```

The number of SCENRED errors are contained in the parameter `ScenRedReport` of the SCENRED output file (if it could be created). The occurence of an error can also be detected from the last line that SCENRED sends to the screen:

```
**** SCENRED ErrCode=...
```

The numerical values of `ErrCode` and their meaning are given below.

ErrCode	Meaning
1	(for internal use)
2	fatal error while reading from SCENRED input file
3	fatal error while writing to SCENRED output file
4	fatal error while reading from SCENRED option file
5	log file cannot be opened
6	a memory allocation error occured
7	there are missing input parameters
8	could not access the GAMS names for the nodes
9	(for internal use)
10	ancestor set not given or contains too many entries
11	node probabilities cannot be not read or are wrong
12	random values for the nodes cannot be read
13	input parameters are out of range
14	ancestor set contains a cycle
15	incomplete scenarios or forests detected
16	fatal error in reduction algorithm (not enough memory)
17	running time limit reached

9 SCENRED Warnings

SCENRED warnings are caused by misspecification of the initial scenarios that can be possibly fixed. When SCENRED encounters such an error in the input files or in the scenario tree, it sends a message to the screen and to the log file. These messages always start with

```
**** SCENRED Warning ...
```

The following list gives an overview of the cases that produce warnings, and the action taken by SCENRED in these cases.

- The user assigned an option value that is out of range.
  Action: Assign the default value.
- Both parameters `red_num_leaves` and `red_percentage` are assigned nontrivial values.
  Action: The value of `red_percentage` will be ignored.
- The scenario probabilities (probabilities of leaves) do not sum up to 1.
  Action: The scenario probabilities are rescaled. Assign new probabilities to the remaining (inner) nodes that are consistent with the scenario probabilities.
- Missing probabilities of inner nodes.
  Action: Assign node probabilities that are consistent with the scenario probabilities.
- The ancestor set contains more than one ancestor for a node.
  Action: SCENRED assumes to be given a successor set instead of an ancestor set (i.e., the transpose of an ancestor matrix. This means that the graph corresponding to the ancestor set has the wrong orientation). SCENRED starts the tree diagnostic for the successor set. The reduced tree will be defined in terms of a successor set as well (if the successor set passes the tree diagnostic and if SCENRED locates no fatal error during the run).
- The fast backward method delivered a result, but the result cannot be improved by the forward or backward method (running time limit reached).
  Action: Use the result of the fast backward method.

SCENRED–2

Contents

1 Introduction

Scenred2 is a fundamental update of the well-known scenario reduction software Scenred. A lot of new features come along with the latest release version. Beside updates and extentions concerning the control of the scenario reduction action an all new device for scenario tree construction has been implemented in Scenred2. Moreover, a lot of visualization functions to plot scenario trees and scenario processes linked to the free Gnuplot plotting software are available with Scenred2 now.

Table: Summary of basic new functions in Scenred2

Description	**Section**
Additional options for controlling the scenario reduction	3
New device of scenario tree construction	4
Visualization of scenario trees and processes	5
Command line interface and data export	6

2 Using Gams/Scenred2

Successful applying Scenred or Scenred2 requires a special formulation of the stochastic programming model within the Gams program. Probabilistic information must be given by a set of nodes implying a certain ancestor structure including a well-defined root node. Note that the usage of Gams/Scenred2 is basically the same as the usage of Gams/Scenred. Hence, it is recommended for new users to look at the Scenred documentation first. All details about how to organize your Gams program, how to run Scenred from the Gams program by using the gdx interface, and, of course, examples can be found in that documentation.

The Gams/Scenred2 link provides the same gdx interface. But, due to new features some small changes in controlling the options are needed. Scenred2 supports now two types of option files. The first one is the SR-Command-File which must be passed to Scenred2 together with the Scenred2 call. The second one, the SR-Option-File includes more specific options to control the selected scenario reduction or scenario construction methods and can be declared in the SR-Command-File.

SR-Command-File

The command file includes the basic specifications. These are input/output gdx file names, the log file name, all other file names which are needed for diverse visualization and output options. It also includes the name of the SR-Option-File.

Table: Supported options of SR-Command-File

Option	Description	Required
`log_file`	specify a log file name	yes
`input_gdx`	specify the gdx input file for Scenred	yes
`output_gdx`	specify the gdx output file of Scenred	yes
`sr_option`	specify a SR-Option-File	no
`visual_init`	specify a name for visualization of input tree	no
`visual_red`	specify a name for visualization of reduced/constructed tree	no
`plot_scen`	specify a name for visualization of scenario processes	no
`out_scen`	specify a file for scenario data output in fan format	no
`out_tree`	specify a file for scenario data output in tree format	no

Example:

Scenred2 must be called together with a command file, which contains at least all required options. The data exchange via the gdx interface and the Scenred2 call from the Gams program is of the form (be careful with the meanings and right order of gdx symbols):

```
execute_unload 'srin.gdx', ScenRedParms, n, ancestor, prob, random;
execute 'scenred2 scenred.cmd';
execute_load 'srout.gdx', ScenRedReport, ancestor=red_ancestor, prob=red_prob;
```

For example, the command file could be the following (note the compatible gdx file names):

```
* scenred command file 'scenred.cmd'

log_file    sr.log
input_gdx   srin.gdx
output_gdx  srout.gdx
sr_option   scenred.opt
visual_red  tree
out_scen    raw.dat
```

ScenRedParms

With the symbol list of the parameter ScenRedParms and the SR-Option-File all necessary information regarding the Scenred2 run can be assigned. The Gams parameter ScenRedParms can easily included to the Gams program by the statement:

```
$libinclude scenred2
```

Of course, the include must be stated before calling Scenred2. After that statement all supported parameters can be assigned, but at least all required parameters regarding the input scenarios. By the symbols of the parameter ScenRedParms you make also the decision of what features you exactly want to use with Scenred2. Moreover, some other usefull parameters for the Scenred2 run are included in the symbol list of the parameter ScenRedParms.

Table: Supported Scenred2 parameters in ScenRedParms

Symbol	Description	Required
`num_time_steps`	path length from root to leaf	yes
`num_leaves`	leaves/scenarios in the initial tree	yes
`num_nodes`	nodes in the initial tree	yes
`num_random`	random variables assigned to a scenario or node	yes
`red_num_leaves`	desired number of preserved scenarios or leaves	no
`red_percentage`	desired relative distance (accuracy)	no
`reduction_method`	desired reduction method	no
`construction_method`	desired tree construction method	no
`num_stages`	number stages	no
`run_time_limit`	time limit in seconds	no
`report_level`	report level: more messages by higher values	no
`scen_red`	scenario reduction command	no
`tree_con`	tree construction command	no
`visual_init`	visualization initial tree	no
`visual_red`	visualization reduced (constructed) tree	no
`plot_scen`	visualization scenario processes	no
`out_scen`	output of scenario raw data	no
`out_tree`	output of scenario tree data	no

To enable some options assign a value to the parameter. A parameter value of zero (default) disables an option. Note that when running Scenred2 either scenario reduction or scenario tree construction can be performed. Hence, only `scen_red` or `tree_con` should be used at once.

Example:

The following statements describe a possible example setup for proceeding the scenario tree construction with visualization of the scenario tree and output of the scenarios to a raw data file afterwards. Note that for the visualization and the scenario output the name of output files must be specified in the SR-Command-File. Otherwise a warning will inform you about not selected file names.

```
* general parameters

ScenRedParms('num_leaves') = 100;
ScenRedParms('num_nodes') = 200;
ScenRedParms('num_time_steps') = 5;
ScenRedParms('num_random') = 2;
ScenRedParms('report_level') = 2;
ScenRedParms('run_time_limit') = 30;

* execution commands

ScenRedParms('tree_con') = 1;
ScenRedParms('visual_red') = 1;
ScenRedParms('out_scen') = 1;
```

Scenred2 can also be used for plotting tasks only. Disable both the `scen_red` and `tree_con` option and use one ore more visualization options only (see also Section 5 for more details regarding visualizations).

SR-Option-File

The SR-Option-File is the more specific option file and will be passed to Scenred2 by the `sr_option` statement specified in the SR-Command-File. It serves as control unit for available methods provided by Scenred2. The supported options depend on what kind of method is called with Scenred2. A detailed list of all options together with examples are given below for both the scenario reduction and the scenario construction devices (see Sections 3 and 4, respectively). Note that certain parameters can be assigned by using both ScenRedParms and the SR-Option-File. In case of having parameters defined twice a warning by Scenred2 will generated to inform you.

3 Scenario Reduction

The scenario reduction device consists of approved methods for reducing the model size by reducing the number of scenarios in an optimal way. Here it doesn't make any difference whether the input data is structured as scenario tree or not. But note, the classical scenario reduction approach is actually developed for two-stage models. Extensions for the multistage case are planed in the near future. To learn more about the mathematical theory see recent publications, for example [5, 4, 2].

With Scenred2 the most popular and accurate reduction algorithms of forward and backward type are maintained further on. New options make it possible to proceed with the scenario reduction more individual. The most important new parameter is given by the option `metric_type` which allows to control the reduction process by different type of probability distances. Altogether three distances can be selected (see Table below). All probability distances are associated with a special order specification which can be set by the new option `order`. Both options replace the old option `where_random` which is not used any longer.

Table: SR Options – Scenario Reduction

Option	Description
`red_num_leaves`	desired number of scenarios (integer)
`red_percentage`	relative accuracy (number from 0.0 to 1.0)
`reduction_method`	1 - Forward, 2 - Backward, 0 - Default
`metric_type`	1 - Transport (default), 2 - Fortet-Mourier, 3 - Wasserstein
`p_norm`	choice of norm (example: 0 - max, 1 - sum, 2 - Euclidian)
`scaling`	0 - scaling off, 1 - scaling on (default)
`order`	metric order (integer, default is 1)

Example:

For example, a valid SR-Option-File is the following:

```
* scenred option file

reduction_method  1
red_percentage    0.3
metric_type       2
order             2
p_norm            1
scaling           0
```

Lines starting with the star symbol (route symbol can be used too) provide comment lines. The star symbol can also be used to out comment and disable certain options.

4 Scenario Tree Construction

Scenario tree construction is the outstanding all new device of Scenred2. It allows to construct scenario trees as accurate input for multistage stochastic programs (cf. [3]). The input are individual scenarios in form of a fan which must be allocated before calling Scenred2. A lot of options are offered to control the tree construction process. Note that in some cases due to sensibility of certain parameters some tuning is indispensable for producing good results.

Table: SR Options (basic) – Scenario Tree Construction

Option	**Description**
`construction_method`	1 - forward, 2 - backward
`reduction_method`	1 - forward, 2 - backward, used within the iteration
`first_branch`	time period of first branch (integer)
`red_percentage`	relative accuracy (level from 0.0 to 1.0)
`eps_growth`	1 - linear, 2 - exponential
`eps_evolution`	tree structure parameter (from 0.0 to 1.0)
`scaling`	0 - scaling off, 1 - scaling on (default)
`order`	order of metric

The Table above displays the main options to control the tree construction process. They are very simular to the reduction options. The role of the option `red_percentage` is here to prescribe a total epsilon accuracy (level) for the approximation scheme. But the approximation scheme is based on stagewise approximations which requires a splitting of the total level to the stages. Two strategies are offered by Scenred2 a linear and an exponential mapping of the total level to the intermediate levels. Use option `eps_growth` to select one of them. Both strategies allow a second tuning parameter `eps_evolution` which effects the slope of the epsilon splitting.

Even though this kind of control may generate good results for many applications, sometimes a more individual control can be needed. For example, some applications require a localization of branching stages. Moreover, to setup approximation bounds directly to stages can be very useful. To this end the standard options are extended by a new section environment.

Additional options – The section environment

An alternative control for the accurate constructions is provided by using the section environment. The section environment aims to establish a better monitoring of the construction process. There are overall three section types supported by Scenred2 with the same syntax.

Branching control:

This section environment allows to specify branching points, i.e., an explicit selection of stages serving for branching. For example, use

```
section branching
  2
  4
  6
end
```

to allow branching only at time period 2, 4, and 6. Note that each stage statement must be placed in one line. But stages can be merged. A shorter formulation of the same contents can be written in closed form

```
section branching
```

```
  2*6  2
end
```

This statement reads branching within time periods from period 2 to period 6 with increment 2 steps. Both assignments can be combined and should be used together with the `red_percentage` option.

Epsilon control:

In the simular way by the epsilon section it is possible to assign epsilon tolerances for the stage approximations explicitly. This environment overcomes difficulties at times coming across with the automatic epsilon control. Note that this environment disables the option `red_percentage`. For example, use

```
section epsilon
  2    0.04
  3*4  0.03
  5    0.02
  6    0.01
end
```

to control the approximation scheme by assigning different epsilon values per stage. Note that the value 0.03 is assigned to time period 3 and 4 in the example.

Node control:

The node control is the most specific control you have over the tree construction. With this environment the number of nodes of the tree which will generated can be assigned for each time stage explicitly. For example, use

```
section num_nodes
  1    1
  2*3  5
  4*5  10
  6    15
end
```

The syntax is the same as before. Note that only one section environment can be use at once. In particular, only the first section environment detected in the option file is used. The section environment can be out commented like a standard option too.

Experimental option:

There is one other useful option to speed up computations when building different scenario trees from exactly the same input data. In this case the scenario distances needed to compute the trees could be saved to a external file at the first run and reloaded at later runs. Hence, the distances must be calculated only once. For example, use the option

```
write_distance  dist.sr2
```

to save the computed distances to the file 'dist.sr2'. To reload them at next run use the option

```
read_distance  dist.sr2
```

The option is classified as experimental since no validation of the input file takes place. Before using this option, please ensure that the distances loaded with the `read_distance` option are the right ones.

Example:

Finally, look at the following example to see a valid SR-Option-File which can be passed to the scenario tree construction:

```
* tree construction option file

construction_method  2
reduction_method     1
order                1
scaling              0

section epsilon
  2*4    0.1
  5      0.2
  6      0.1
end
```

Example problem 'srpchase.gms'

A small example problem has been included to the GAMS Model Library. The implementation can be found in the Gams program 'srpchase.gms'. It might help you to practice in building scenario trees using Gams/Scenred2. The problem is to solve a simple stochastic purchase problem involving three stages. Sample scenarios which are generated from a fixed distribution using a random value generator serve as input for the tree construction.

5 Visualization

Visualization is another all new feature of Scenred2. In this section an easy way for making plots of scenario processes and tree structures is described. To this end you need the free Gnuplot software or any other plotting software which allows plotting directly from simple data files.

The concept of plotting tasks is the following. For each plot two files are generated, a Gnuplot access file (name.plt) and a raw data file (name.dat). The access file contains basic Gnuplot options and it can be adjusted for individual liking afterwards. The default output is the display. The supported plotting commands are

```
visual_init, visual_red, plot_scen
```

for plotting the initial tree structure, the reduced/constructed tree structure, and the scenario process(es), respectively.

Example:

For example, to visualize the constructed tree use the option

```
visual_red tree
```

within the SR-Command-File to specify the name for the output and activate the ScenRedParms parameter

```
ScenRedParms('visual_red') = 1;
```

in the Gams program. The result are the output files 'tree.plt' and 'tree.dat'. To compute the picture now you simply open the file 'tree.plt' with Gnuplot from the directory, where both output files are located (that should be the working directory). Alternatively, from the command line prompt call

```
>gnuplot tree.plt
```

Gnuplot will automatically generate the picture. Feel free to change any option in the Gnuplot access file for individual requirements. See also the Gnuplot manual for more details. In particular, to compute a well-scaled encapsulated postscript picture (eps), you simply have to uncomment a few lines in the Gnuplot option file above and to open it with Gnuplot once again.

With the command `plot_scen` the scenario process(es) can be visualized. Note that Scenred2 generates Gnuplot access and data files according to the number of random values.

6 Command Line Interface

The command line interface allows to run Scenred2 stand alone without using Gams. In this case the input for scenario reduction and scenario tree construction is handled by special input data files. The command file will be extended by the parameters having with the ScenRedParms otherwise.

To execute Scenred2 from the command line prompt together with a specified command file (which is required again), for example, call

```
>scenred2 command.file -nogams
```

To avoid diverse error messages do not forget the '-nogams' option to switch off the Gams interface. The command file can include some of the following options.

```
   report_level  <integer>
   runtime_limit <integer>
   read_scen     <input file>
   scen_red      <option file>
   tree_con      <option file>
   visual_init   <name>
   visual_red    <name>
   plot_scen     <name>
   out_scen      <file name>
   out_tree      <file name>
```

The denotation is not accidental the same as in case of using the Gams interface. The meaning of a certain option is maintained for the command line interface. To compute any scenario reduction or scenario tree construction the same SR-Option-Files are supported. It remains to clarify the data input format which comes across with the new `read_scen` command.

Data input format

To feed Scenred2 with data the scenario parameters must be passed by the `read_scen` command. Two types of input file formats are accepted.

a) The tree format:

This file is a formated data file including all information of the input scenarios tree. It must have a header with dimension information and the scenario data separated for each node. The header includes the type declaration, the number of nodes, and the number of random values.

The data part starts with the key word `DATA` (do not forget). The tree data has to be ordered node by node. For every node the following information is expected separated by white spaces: The unique predecessor node (root node points to itself) followed by the node probability and followed by the assigned number of random data values.

All information to one node should be written to one line (only for clearness reasons). Comment lines are allowed.

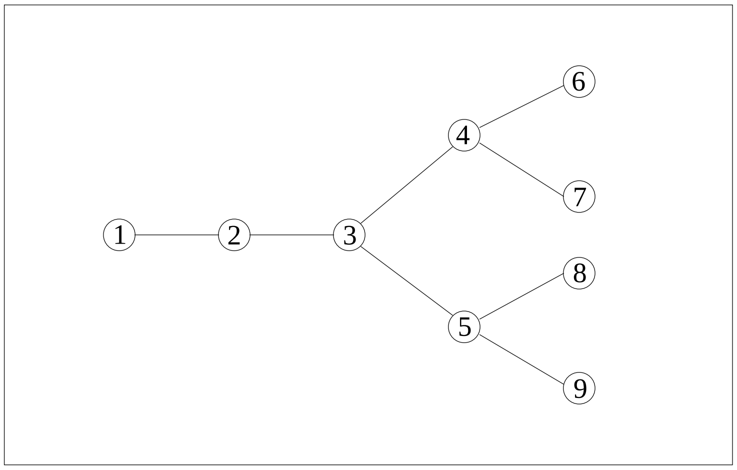
Match the following conventions:

– Nodes are identified by a sequence of integer numbers.
– The root node is expected to be the node '1'.
– The predecessor of root is '1' too, i.e., the root points to itself.
– All nodes numbers require a canonical order by stages and scenarios (see example).

<u>Example:</u>

```
# input tree example for scenred

TYPE TREE

NODES   9
RANDOM  4

DATA
* PRED PROB  RAND-1 RAND-2 RAND-3 RAND-4
   1    1.0    42.5    9.1    7.5  120.0
   1    1.0    39.8   11.2    8.4   90.0
   2    1.0    37.6   14.0    6.3  110.0
   3    0.5    38.9   12.4    8.1  130.0
   3    0.5    35.7   13.8    7.5  120.0
   4   0.25    40.3   14.9    7.2  120.0
   4   0.25    38.4   15.2    8.9  100.0
   5    0.3    37.6   14.9    9.3   80.0
   5    0.2    36.3   12.8   10.3   90.0
END
```

Figure: The scenario structure of the example tree



b) The fan format:

A scenario fan serves as input for the scenario tree construction but it can be used also for the scenario reduction. The scenario fan represents a special form of a scenario tree, where we consider individual scenarios merged to a collective root node (the root node can also be viewed here as some kind of artificial node).

Accordingly, the fan input file is a formated data file including all information of the scenarios in individual form now. It must have a simular header with dimension information and the scenario data separated now for each scenario. The header gets the type declaration `FAN` instead of `TREE` and includes the number of scenarios, the number of time periods, and the number of random values. The data part is opened again with the `DATA` key word.

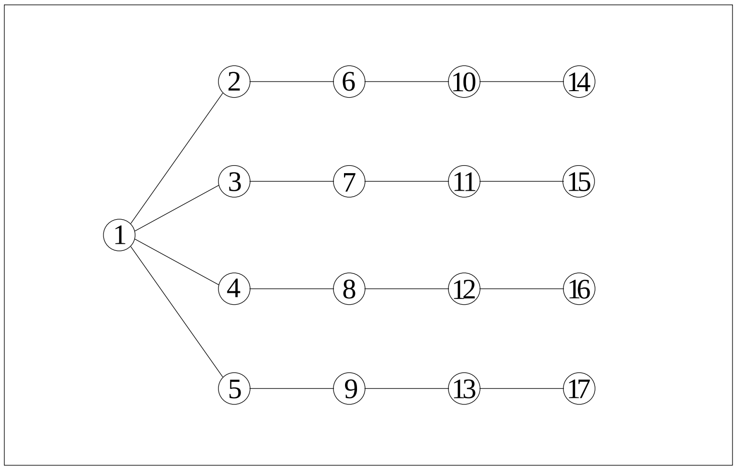
Every scenario is specified by a dataset including the scenario probability first followed by the different random values in ascending order w.r.t. time periods. All entries must be separated by a white space. Comment lines can be placed by the star and route symbols again. Note that in case of having an undetermined root node the mean of random values will taken for the first time period to appoint a unique root node. The example tree represented as input in scenario fan format is displayed in the next example.

Example:

```
# input fan example for scenred

TYPE  FAN

TIME    5
SCEN    4
RANDOM  4

DATA
0.2500
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
38.9     12.4      8.1     130.0
40.3     14.9      7.2     120.0

0.2500
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
38.9     12.4      8.1     130.0
38.4     15.2      8.9     100.0

0.3000
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
35.7     13.8      7.5     120.0
37.6     14.9      9.3      80.0

0.2000
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
35.7     13.8      7.5     120.0
36.3     12.8     10.3      90.0
END
```

Note that even though all scenarios coincide at the first three time periods, in this example, the scenarios will represented by one node each for every time period by the fan input format. The exception is the first time period, where a unique root node is expected in general and, therefore, only one node is assigned. The following picture shows the structure of the scenario fan which is generated by the example input.

Figure: The structure of the example input in fan format



Data Export

Scenred2 allows to export scenario data after computing the scenario reduction or scenario tree construction to external data files. Data export is available for both the Gams and the command line interface. To export data from Scenred2 two output options `out_tree` and `out_scen` can be use. These options generate data files according to the tree and fan format, respectively. The name of the data files will be specified in the SR-Command-File. When using the Gams interface the options must be connected by activating the corresponding ScenRedParms parameter, additionally.

7 A Simplified Interface to Scenred2: `$libinclude runscenred2`

While the previously described interface between GAMS and Scenred2 provides a maximum of flexibility, it also is rather complex and error-prone. The GAMS utility `runscenred2` tries to hide most of the mechanics of the GAMS/Scenred2 interface. The call to `runscenred2` looks as follows:

```
$libinclude runscenred2 myprefix tree_con n tree p rtree rp rv1 rv2
```

Table: runscenred2 Arguments:

	Argument	Description
1	`myprefix`	base name for files used with Scenred2
2	`tree_con` or `scen_red`	select Scenred2 action: tree construction or scenario reduction
3	`n`	the set of nodes in the tree
4	`tree`	the set of ancestor relations describing the tree
5	`p`	the parameter containing the node probablities
6	`rtree`	the set of ancestor relations of the reduced tree (output)
7	`rp`	the parameter containing the node probablities for the reduced tree (output)
8-	`rv1`, `rv2`, ...	parameters containing random values of the nodes

The table above describes the arguments of the `runscenred2` call. Arguments 3, 4, 5, 8 and following correspond to the symbols that need to be exported to the Scenred2 data input file (done with the `execute_unload` call in the complex interface). The output arguments 6 and 7 correspond to the symbols imported from te Scenred2 data output file (done with the `execute_load` call in the complex interface). The parameters `ScenRedParms` and `ScenRedReport` are invisibly communicated with Scenred2.

The second argument instructs Scenred2 either to construct a tree (`tree_con`) or to reduce a tree (`scen_red`).

Instead of providing an explicit name for all the different files in the Scenred2 command file, the first argument determines the name of all files using a simple naming scheme. The following name scheme is observed:

Filename	**Command option**	**Description**
`sr2`*myprefix*`.log`	`log_file`	log file name
`sr2`*myprefix*`_in.gdx`	`input_gdx`	gdx input file name
`sr2`*myprefix*`_out.gdx`	`output_gdx`	gdx output file name
`sr2`*myprefix*`.opt`	`sr_option`	option file name
`sr2`*myprefix*`_vi.plt`	`visual_init`	file name for visualization of input tree
`sr2`*myprefix*`_vr.plt`	`visual_red`	file name for visualization of reduced/constructed tree
`sr2`*myprefix*`_plot.plt`	`plot_scen`	file name for visualization of scenario process
`sr2`*myprefix*`_raw.dat`	`out_scen`	file name for scenario data output in fan format
`sr2`*myprefix*`_tree.dat`	`out_tree`	file name for scenario data output in tree format

The first three files (`log_file`, `input_gdx` and `output_gdx`) are always used. The only optional input file `sr_option` is read by Scenred2 if `ScenRedParms('sroption')=1`. When you create this file, make sure to use the proper file name. The output files are created by Scenred2 if the corresponding option is set to 1 in `ScenRedParms`, e.g. `ScenRedParms('out_tree')=1`.

In addition to a simpler communication of data between GAMS and Scenred2, the newer versions of GAMS/Scenred2 (starting with GAMS distribution 23.1) release the user of setting required fields in the `ScenRedParms` parameter: `num_time_steps`, `num_leaves`, `num_nodes`, and `num_random`. GAMS/Scenred2 calculates these numbers from its input data. In case the user still sets these fields, Scenred2 will ensure that the internbally calculated numbers and the user provided numbers match.

References

I Heitsch, H.: Stabilität und Approximation stochastischer Optimierungsprobleme, dissertation, Logos Verlag Berlin, 2007.

II Heitsch, H.; Römisch, W.: Scenario tree reduction for multistage stochastic programs, *Computational Management Science* 6 (2009), 117–133.

III Heitsch, H.; Römisch, W.: Scenario tree modeling for multistage stochastic programs, *Mathematical Programming* 118 (2009), 371–406.

IV Heitsch, H.; Römisch, W.; Strugarek, C.: Stability of multistage stochastic programs, *SIAM Journal on Optimization* 17 (2006), 511–525.

V Heitsch, H.; Römisch, W.: A note on scenario reduction for two-stage stochastic programs, *Operations Research Letters* 35 (2007), 731–738 .

VI Heitsch, H.; Römisch, W.: Scenario reduction algorithms in stochastic programming. *Computational Optimization and Applications* 24 (2003), 187–206.

SCIP

Stefan Vigerske, Humboldt University Berlin, Germany

Contents

1 Introduction

SCIP (**S**olving **C**onstraint **I**nteger **P**rograms) is developed at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). The SCIP main developer had been Tobias Achterberg, current developers are Timo Berthold, Gerald Gamrath, Stefan Heinz, Gregor Hendel, Thorsten Koch, Stefan Vigerske, Robert Waniek, Michael Winkler, and Kati Wolter. Since SCIP is distributed under the ZIB Academic License, it is only available for users with a **GAMSacademic 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 solver or as a framework for branch-cut-and-price. Within GAMS, the MIP and MIQCP solving facilities of SCIP are available.

For more detailed information, we refer to [1, 2, 3, 4, 5, 7] and the SCIP web site `http://scip.zib.de`, especially the list of papers listed at `http://scip.zib.de/related.shtml`.

GAMS/SCIP uses the linear solver SoPlex [8] as LP solver and the COIN-OR Interior Point Optimizer Ipopt [6] as nonlinear solver.

2 Model requirements

SCIP supports continuous, binary, integer, semi-continuous, and semi-integer variables, special ordered sets, and branching priorities. Indicator constraints are not supported by the interface yet.

3 Usage

The following statement can be used inside your GAMSprogram to specify using SCIP

```
  Option MIP = SCIP;     { or QCP or RMIQCP or MIQCP }
```

The above statement should appear before the Solve statement. If SCIP was specified as the default solver during GAMSinstallation, the above statement is not necessary.

If a continuous linear program (LP or RMIP) is given to GAMS/SCIP, the linear programming solver is called directly. In this case, it is not possible to provide an option file.

GAMS/SCIP currently does not support the GAMSBranch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/SCIP with BCH, please consider to use a GAMSsystem of version ≤ 23.3, available at http://www.gams.com/download/download_old.htm.

3.0.1 Specification of SCIP Options

GAMS/SCIP currently supports the GAMSparameters `reslim`, `iterlim`, `nodlim`, `optcr`, and `optca`. Further, under Linux and Windows, the option `threads` can be used to control the number of threads used in the linear algebra routines of Ipopt.

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:

```
presolving/probing/maxrounds = 0
separating/maxrounds         = 0
separating/maxroundsroot     = 0
```

It causes GAMS/SCIP to disable probing during presolve and to turn off all cut generators.

4 Detailed Options Description

SCIP supports a large set of options. Sample option files can be obtained from
http://www.gams.com/~svigerske/scip2.0.

In the following we give a detailed list of some SCIP options. A list of all SCIP options can be obtained from
http://scip.zib.de/doc/html/PARAMETERS.html.

GAMS interface specific options

`gams/names` (boolean) FALSE
This option causes GAMS names for the variables and equations to be loaded into SCIP. These names will then be used for error messages, log entries, and so forth. Turning names off may help if memory is very tight.

`gams/mipstart` (boolean) TRUE
This option controls the use of advanced starting values for mixed integer programs. A setting of TRUE indicates that the variable level values should be checked to see if they provide an integer feasible solution before starting optimization.

`gams/print_statistics` (boolean) FALSE
This option controls the printing of solve statistics after a MIP solve. Turning on this option indicates that statistics like the number of generated cuts of each type or the calls of heuristics are printed after the MIP solve.

`gams/interactive` (boolean) FALSE
whether a SCIP shell should be opened instead of issuing a solve command (this option is not available in demo mode)

Branching

`branching/preferbinary` (boolean) FALSE
should branching on binary variables be preferred?

`branching/clamp` ($0 \leq$ real ≤ 0.5) 0.2
minimal relative distance of branching point to bounds when branching on a continuous variable

`branching/allfullstrong/priority` ($-536870912 \leq$ integer ≤ 536870911) -1000
priority of branching rule <allfullstrong>

`branching/allfullstrong/maxdepth` ($-1 \leq$ integer) -1
maximal depth level, up to which branching rule <allfullstrong> should be used (-1 for no limit)

`branching/allfullstrong/maxbounddist` ($0 \leq$ real ≤ 1) 1
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)

`branching/fullstrong/priority` ($-536870912 \leq$ integer ≤ 536870911) 0
priority of branching rule <fullstrong>

`branching/fullstrong/maxdepth` ($-1 \leq$ integer) -1
maximal depth level, up to which branching rule <fullstrong> should be used (-1 for no limit)

`branching/fullstrong/maxbounddist` ($0 \leq$ real ≤ 1) 1
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)

`branching/inference/priority` ($-536870912 \leq$ integer ≤ 536870911) 1000
priority of branching rule <inference>

`branching/inference/maxdepth` ($-1 \leq$ integer) -1
maximal depth level, up to which branching rule <inference> should be used (-1 for no limit)

`branching/inference/maxbounddist` ($0 \leq$ real ≤ 1) 1
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)

`branching/inference/useweightedsum` (`boolean`) `TRUE`
should a weighted sum of inference, conflict and cutoff weigths be used?

`branching/mostinf/priority` ($-536870912 \leq$ integer ≤ 536870911) 100
priority of branching rule <mostinf>

`branching/mostinf/maxdepth` ($-1 \leq$ integer) -1
maximal depth level, up to which branching rule <mostinf> should be used (-1 for no limit)

`branching/mostinf/maxbounddist` ($0 \leq$ real ≤ 1) 1
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)

`branching/leastinf/priority` ($-536870912 \leq$ integer ≤ 536870911) 50
priority of branching rule <leastinf>

`branching/leastinf/maxdepth` ($-1 \leq$ integer) -1
maximal depth level, up to which branching rule <leastinf> should be used (-1 for no limit)

`branching/leastinf/maxbounddist` ($0 \leq$ real ≤ 1) 1
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)

`branching/pscost/priority` ($-536870912 \leq$ integer ≤ 536870911) 2000
priority of branching rule <pscost>

`branching/pscost/maxdepth` ($-1 \leq$ integer) -1
maximal depth level, up to which branching rule <pscost> should be used (-1 for no limit)

`branching/pscost/maxbounddist` ($0 \leq$ real ≤ 1) 1
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)

`branching/pscost/strategy` (`character`) `r`
strategy for computing score of external branching candidates (b: rb-int-br, r: rb-int-br-rev, i: rb-inf)

`branching/random/priority` $(-536870912 \leq \text{integer} \leq 536870911)$ -100000
priority of branching rule <random>

`branching/random/maxdepth` $(-1 \leq \text{integer})$ -1
maximal depth level, up to which branching rule <random> should be used (-1 for no limit)

`branching/random/maxbounddist` $(0 \leq \text{real} \leq 1)$ 1
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)

`branching/random/seed` $(0 \leq \text{integer})$ 0
initial random seed value

`branching/relpscost/priority` $(-536870912 \leq \text{integer} \leq 536870911)$ 10000
priority of branching rule <relpscost>

`branching/relpscost/maxdepth` $(-1 \leq \text{integer})$ -1
maximal depth level, up to which branching rule <relpscost> should be used (-1 for no limit)

`branching/relpscost/maxbounddist` $(0 \leq \text{real} \leq 1)$ 1
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)

`branching/relpscost/sbiterquot` $(0 \leq \text{real})$ 0.5
maximal fraction of strong branching LP iterations compared to node relaxation LP iterations

`branching/relpscost/sbiterofs` $(0 \leq \text{integer})$ 100000
additional number of allowed strong branching LP iterations

`branching/relpscost/initcand` $(0 \leq \text{integer})$ 100
maximal number of candidates initialized with strong branching per node

`branching/relpscost/inititer` $(0 \leq \text{integer})$ 0
iteration limit for strong branching initializations of pseudo cost entries (0: auto)

Conflict analysis

`conflict/enable` (`boolean`) `TRUE`
should conflict analysis be enabled?

`conflict/useprop` (`boolean`) `TRUE`
should propagation conflict analysis be used?

`conflict/useinflp` (`boolean`) `TRUE`
should infeasible LP conflict analysis be used?

`conflict/useboundlp` (`boolean`) `FALSE`
should bound exceeding LP conflict analysis be used?

`conflict/usesb` (`boolean`) `FALSE`
should infeasible/bound exceeding strong branching conflict analysis be used?

`conflict/usepseudo` (`boolean`) `TRUE`
should pseudo solution conflict analysis be used?

`conflict/preferbinary` (`boolean`) `FALSE`
should binary conflicts be preferred?

`conflict/restartnum` $(0 \leq \text{integer})$ 0
number of successful conflict analysis calls that trigger a restart (0: disable conflict restarts)

`conflict/restartfac` $(0 \leq \text{real})$ 1.5
factor to increase restartnum with after each restart

Constraints

`constraints/linear/sepafreq` $(-1 \leq \text{integer})$ 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/linear/propfreq` ($-1 \leq$ integer) frequency for propagating domains (-1: never, 0: only in root node)	1
`constraints/linear/maxrounds` ($-1 \leq$ integer) maximal number of separation rounds per node (-1: unlimited)	5
`constraints/linear/maxroundsroot` ($-1 \leq$ integer) maximal number of separation rounds per node in the root node (-1: unlimited)	-1
`constraints/linear/maxsepacuts` ($0 \leq$ integer) maximal number of cuts separated per separation round	50
`constraints/linear/maxsepacutsroot` ($0 \leq$ integer) maximal number of cuts separated per separation round in the root node	200
`constraints/linear/separateall` (`boolean`) should all constraints be subject to cardinality cut generation instead of only the ones with non-zero dual value?	`FALSE`
`constraints/and/sepafreq` ($-1 \leq$ integer) frequency for separating cuts (-1: never, 0: only in root node)	1
`constraints/and/propfreq` ($-1 \leq$ integer) frequency for propagating domains (-1: never, 0: only in root node)	1
`constraints/bounddisjunction/sepafreq` ($-1 \leq$ integer) frequency for separating cuts (-1: never, 0: only in root node)	-1
`constraints/bounddisjunction/propfreq` ($-1 \leq$ integer) frequency for propagating domains (-1: never, 0: only in root node)	1
`constraints/conjunction/sepafreq` ($-1 \leq$ integer) frequency for separating cuts (-1: never, 0: only in root node)	-1
`constraints/conjunction/propfreq` ($-1 \leq$ integer) frequency for propagating domains (-1: never, 0: only in root node)	-1
`constraints/countsols/sepafreq` ($-1 \leq$ integer) frequency for separating cuts (-1: never, 0: only in root node)	-1
`constraints/countsols/propfreq` ($-1 \leq$ integer) frequency for propagating domains (-1: never, 0: only in root node)	-1
`constraints/countsols/active` (`boolean`) is the constraint handler active?	`FALSE`
`constraints/countsols/sparsetest` (`boolean`) should the sparse solution test be turned on?	`TRUE`
`constraints/countsols/discardsols` (`boolean`) is it allowed to discard solutions?	`TRUE`
`constraints/countsols/collect` (`boolean`) should the solutions be collected?	`FALSE`
`constraints/countsols/sollimit` ($-1 \leq$ integer ≤ -1) counting stops, if the given number of solutions were found (-1: no limit)	-1
`constraints/cumulative/sepafreq` ($-1 \leq$ integer) frequency for separating cuts (-1: never, 0: only in root node)	1
`constraints/cumulative/propfreq` ($-1 \leq$ integer) frequency for propagating domains (-1: never, 0: only in root node)	5
`constraints/cumulative/usebinvars` (`boolean`) should the binary representation be used?	`FALSE`
`constraints/cumulative/usecoretimes` (`boolean`) should coretimes be propagated?	`TRUE`

`constraints/cumulative/usecoretimesholes` (boolean) FALSE
should coretimes be propagated to detect holes?

`constraints/cumulative/localcuts` (boolean) FALSE
should cuts be added only locally?

`constraints/cumulative/usecovercuts` (boolean) TRUE
should covering cuts be added every node?

`constraints/cumulative/useedgefinding` (boolean) FALSE
should edge finding be used?

`constraints/cumulative/useenergeticreasoning` (boolean) FALSE
should energetic reasoning be used?

`constraints/cumulative/cutsasconss` (boolean) TRUE
should the cumulative constraint create cuts as knapsack constraints?

`constraints/indicator/sepafreq` ($-1 \leq$ integer) 10
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/indicator/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/integral/sepafreq` ($-1 \leq$ integer) -1
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/integral/propfreq` ($-1 \leq$ integer) -1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/knapsack/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/knapsack/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/linear/upgrade/knapsack` (boolean) TRUE
enable linear upgrading for constraint handler <knapsack>

`constraints/knapsack/maxrounds` ($-1 \leq$ integer) 5
maximal number of separation rounds per node (-1: unlimited)

`constraints/knapsack/maxroundsroot` ($-1 \leq$ integer) -1
maximal number of separation rounds per node in the root node (-1: unlimited)

`constraints/knapsack/maxsepacuts` ($0 \leq$ integer) 50
maximal number of cuts separated per separation round

`constraints/knapsack/maxsepacutsroot` ($0 \leq$ integer) 200
maximal number of cuts separated per separation round in the root node

`constraints/linking/sepafreq` ($-1 \leq$ integer) 1
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/linking/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/linking/linearize` (boolean) FALSE
this constraint will not propagate or separate, linear and setppc are used?

`constraints/logicor/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/logicor/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/linear/upgrade/logicor` (boolean) TRUE
enable linear upgrading for constraint handler <logicor>

`constraints/or/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/or/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/orbitope/sepafreq` ($-1 \leq$ integer) 5
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/orbitope/propfreq` ($-1 \leq$ integer) -1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/quadratic/sepafreq` ($-1 \leq$ integer) 2
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/quadratic/propfreq` ($-1 \leq$ integer) 10
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/quadratic/replacebinaryprod` ($0 \leq$ integer) ∞
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)

`constraints/quadratic/empathy4and` ($0 \leq$ integer ≤ 2) 0
empathy level for using the AND constraint handler: 0 always avoid using AND; 1 use AND sometimes; 2 use AND as often as possible

`constraints/quadratic/minefficacysepa` ($0 \leq$ real) 0.0001
minimal efficacy for a cut to be added to the LP during separation; overwrites separating/efficacy

`constraints/quadratic/minefficacyenfo` ($0 \leq$ real) $2 \cdot 10^{-6}$
minimal target efficacy of a cut in order to add it to relaxation during enforcement (may be ignored)

`constraints/quadratic/scaling` (`boolean`) `TRUE`
whether a quadratic constraint should be scaled w.r.t. the current gradient norm when checking for feasibility

`constraints/quadratic/cutmaxrange` ($0 \leq$ real) 10^{10}
maximal range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation

`constraints/quadratic/linearizenlpsol` (`boolean`) `TRUE`
whether convex quadratic constraints should be linearized in a solution found by the NLP or RENS heuristic

`constraints/quadratic/checkcurvature` (`boolean`) `TRUE`
whether multivariate quadratic functions should be checked for convexity/concavity

`constraints/quadratic/linfeasshift` (`boolean`) `TRUE`
whether to try to make solutions in check function feasible by shifting a linear variable (esp. useful if constraint was actually objective function)

`constraints/setppc/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/setppc/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/linear/upgrade/setppc` (`boolean`) `TRUE`
enable linear upgrading for constraint handler <setppc>

`constraints/SOS1/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/SOS1/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/SOS1/branchSOS` (`boolean`) `TRUE`
Use SOS1 branching in enforcing (otherwise leave decision to branching rules)?

`constraints/SOS1/branchNonzeros` (`boolean`) `FALSE`
Branch on SOS constraint with most number of nonzeros?

`constraints/SOS1/branchWeight` (boolean) FALSE
Branch on SOS cons. with highest nonzero-variable weight for branching (needs branchNonzeros = false)?

`constraints/SOS2/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/SOS2/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/soc/sepafreq` ($-1 \leq$ integer) 1
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/soc/propfreq` ($-1 \leq$ integer) 20
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/quadratic/upgrade/soc` (boolean) TRUE
enable quadratic upgrading for constraint handler <soc>

`constraints/soc/nauxvars` ($0 \leq$ integer) 0
number of auxiliary variables to use when creating a linear outer approx. of a SOC3 constraint; 0 to turn off

`constraints/soc/glineur` (boolean) TRUE
whether the Glineur Outer Approximation should be used instead of Ben-Tal Nemirovski

`constraints/soc/linearizenlpsol` (boolean) TRUE
whether SOC constraints should be linearized in a solution found by the NLP or RENS heuristic

`constraints/soc/minefficacy` ($0 \leq$ real) 0.0001
minimal efficacy of a cut to be added to LP in separation

`constraints/soc/linfeasshift` (boolean) TRUE
whether to try to make solutions feasible in check by shifting the variable on the right hand side

`constraints/soc/nlpform` (character) a
which formulation to use when adding a SOC constraint to the NLP (a: automatic, q: nonconvex quadratic form, s: convex sqrt form, e: convex exponential-sqrt form)

`constraints/varbound/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/varbound/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

`constraints/linear/upgrade/varbound` (boolean) TRUE
enable linear upgrading for constraint handler <varbound>

`constraints/xor/sepafreq` ($-1 \leq$ integer) 0
frequency for separating cuts (-1: never, 0: only in root node)

`constraints/xor/propfreq` ($-1 \leq$ integer) 1
frequency for propagating domains (-1: never, 0: only in root node)

Output

`display/verblevel` ($0 \leq$ integer ≤ 5) 4
verbosity level of output

`display/width` ($0 \leq$ integer) 80
maximal number of characters in a node information line

`display/freq` ($-1 \leq$ integer) 100
frequency for displaying node information lines

`display/headerfreq` ($-1 \leq$ integer) 15
frequency for displaying header lines (every n'th node information line)

`display/lpinfo` (boolean) FALSE
should the LP solver display status messages?

`display/sols/active` ($0 \leq$ integer ≤ 2) 0
display activation status of display column <sols> (0: off, 1: auto, 2:on)

`display/feasST/active` ($0 \leq$ integer ≤ 2) 0
display activation status of display column <feasST> (0: off, 1: auto, 2:on)

`display/solfound/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <solfound> (0: off, 1: auto, 2:on)

`display/time/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <time> (0: off, 1: auto, 2:on)

`display/nnodes/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <nnodes> (0: off, 1: auto, 2:on)

`display/nodesleft/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <nodesleft> (0: off, 1: auto, 2:on)

`display/lpiterations/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <lpiterations> (0: off, 1: auto, 2:on)

`display/lpavgiterations/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <lpavgiterations> (0: off, 1: auto, 2:on)

`display/memused/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <memused> (0: off, 1: auto, 2:on)

`display/depth/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <depth> (0: off, 1: auto, 2:on)

`display/maxdepth/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <maxdepth> (0: off, 1: auto, 2:on)

`display/plungedepth/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <plungedepth> (0: off, 1: auto, 2:on)

`display/nfrac/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <nfrac> (0: off, 1: auto, 2:on)

`display/nexternbranchcands/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <nexternbranchcands> (0: off, 1: auto, 2:on)

`display/vars/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <vars> (0: off, 1: auto, 2:on)

`display/conss/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <conss> (0: off, 1: auto, 2:on)

`display/curconss/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <curconss> (0: off, 1: auto, 2:on)

`display/curcols/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <curcols> (0: off, 1: auto, 2:on)

`display/currows/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <currows> (0: off, 1: auto, 2:on)

`display/cuts/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <cuts> (0: off, 1: auto, 2:on)

`display/separounds/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <separounds> (0: off, 1: auto, 2:on)

`display/poolsize/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <poolsize> (0: off, 1: auto, 2:on)

`display/conflicts/active` ($0 \leq$ integer ≤ 2) 1
display activation status of display column <conflicts> (0: off, 1: auto, 2:on)

`display/strongbranchs/active` ($0 \leq$ integer ≤ 2) display activation status of display column <strongbranchs> (0: off, 1: auto, 2:on)	1
`display/lpobj/active` ($0 \leq$ integer ≤ 2) display activation status of display column <lpobj> (0: off, 1: auto, 2:on)	1
`display/curdualbound/active` ($0 \leq$ integer ≤ 2) display activation status of display column <curdualbound> (0: off, 1: auto, 2:on)	1
`display/estimate/active` ($0 \leq$ integer ≤ 2) display activation status of display column <estimate> (0: off, 1: auto, 2:on)	1
`display/avgdualbound/active` ($0 \leq$ integer ≤ 2) display activation status of display column <avgdualbound> (0: off, 1: auto, 2:on)	1
`display/dualbound/active` ($0 \leq$ integer ≤ 2) display activation status of display column <dualbound> (0: off, 1: auto, 2:on)	1
`display/primalbound/active` ($0 \leq$ integer ≤ 2) display activation status of display column <primalbound> (0: off, 1: auto, 2:on)	1
`display/cutoffbound/active` ($0 \leq$ integer ≤ 2) display activation status of display column <cutoffbound> (0: off, 1: auto, 2:on)	1
`display/gap/active` ($0 \leq$ integer ≤ 2) display activation status of display column <gap> (0: off, 1: auto, 2:on)	1
`display/nsols/active` ($0 \leq$ integer ≤ 2) display activation status of display column <nsols> (0: off, 1: auto, 2:on)	1

Heuristics

`heuristics/actconsdiving/freq` ($-1 \leq$ integer) frequency for calling primal heuristic <actconsdiving> (-1: never, 0: only at depth freqofs)	-1
`heuristics/actconsdiving/freqofs` ($0 \leq$ integer) frequency offset for calling primal heuristic <actconsdiving>	5
`heuristics/actconsdiving/maxlpiterquot` ($0 \leq$ real) maximal fraction of diving LP iterations compared to node LP iterations	0.05
`heuristics/actconsdiving/maxlpiterofs` ($0 \leq$ integer) additional number of allowed LP iterations	1000
`heuristics/actconsdiving/backtrack` (`boolean`) use one level of backtracking if infeasibility is encountered?	`TRUE`
`heuristics/coefdiving/freq` ($-1 \leq$ integer) frequency for calling primal heuristic <coefdiving> (-1: never, 0: only at depth freqofs)	10
`heuristics/coefdiving/freqofs` ($0 \leq$ integer) frequency offset for calling primal heuristic <coefdiving>	1
`heuristics/coefdiving/maxlpiterquot` ($0 \leq$ real) maximal fraction of diving LP iterations compared to node LP iterations	0.05
`heuristics/coefdiving/maxlpiterofs` ($0 \leq$ integer) additional number of allowed LP iterations	1000
`heuristics/coefdiving/backtrack` (`boolean`) use one level of backtracking if infeasibility is encountered?	`TRUE`
`heuristics/crossover/freq` ($-1 \leq$ integer) frequency for calling primal heuristic <crossover> (-1: never, 0: only at depth freqofs)	30
`heuristics/crossover/freqofs` ($0 \leq$ integer) frequency offset for calling primal heuristic <crossover>	0
`heuristics/crossover/nodesofs` ($0 \leq$ integer ≤ -1)	500

number of nodes added to the contingent of the total nodes

`heuristics/crossover/nusedsols` ($2 \leq$ integer) number of solutions to be taken into account	3
`heuristics/crossover/nodesquot` ($0 \leq$ real ≤ 1) contingent of sub problem nodes in relation to the number of nodes of the original problem	0.1
`heuristics/crossover/minfixingrate` ($0 \leq$ real ≤ 1) minimum percentage of integer variables that have to be fixed	0.666
`heuristics/dins/freq` ($-1 \leq$ integer) frequency for calling primal heuristic <dins> (-1: never, 0: only at depth freqofs)	-1
`heuristics/dins/freqofs` ($0 \leq$ integer) frequency offset for calling primal heuristic <dins>	0
`heuristics/dins/nodesofs` ($0 \leq$ integer ≤ -1) number of nodes added to the contingent of the total nodes	5000
`heuristics/dins/nodesquot` ($0 \leq$ real ≤ 1) contingent of sub problem nodes in relation to the number of nodes of the original problem	0.05
`heuristics/dins/minnodes` ($0 \leq$ integer ≤ -1) minimum number of nodes required to start the subproblem	500
`heuristics/dins/solnum` ($1 \leq$ integer) number of pool-solutions to be checked for flag array update (for hard fixing of binary variables)	5
`heuristics/dins/neighborhoodsize` ($1 \leq$ integer) radius (using Manhattan metric) of the incumbent's neighborhood to be searched	18
`heuristics/feaspump/freq` ($-1 \leq$ integer) frequency for calling primal heuristic <feaspump> (-1: never, 0: only at depth freqofs)	20
`heuristics/feaspump/freqofs` ($0 \leq$ integer) frequency offset for calling primal heuristic <feaspump>	0
`heuristics/feaspump/maxlpiterquot` ($0 \leq$ real) maximal fraction of diving LP iterations compared to node LP iterations	0.01
`heuristics/feaspump/objfactor` ($0 \leq$ real ≤ 1) factor by which the regard of the objective is decreased in each round, 1.0 for dynamic	1
`heuristics/feaspump/alphadiff` ($0 \leq$ real ≤ 1) threshold difference for the convex parameter to perform perturbation	1
`heuristics/feaspump/maxlpiterofs` ($0 \leq$ integer) additional number of allowed LP iterations	1000
`heuristics/feaspump/neighborhoodsize` ($1 \leq$ integer) radius (using Manhattan metric) of the neighborhood to be searched in stage 3	18
`heuristics/feaspump/beforecuts` (`boolean`) should the feasibility pump be called at root node before cut separation?	`TRUE`
`heuristics/feaspump2/usefp20` (`boolean`) should an iterative round-and-propagate scheme be used to find the integral points?	`FALSE`
`heuristics/feaspump2/pertsolfound` (`boolean`) should a random perturbation be performed if a feasible solution was found?	`TRUE`
`heuristics/feaspump2/stage3` (`boolean`) should we solve a local branching sub-MIP if no solution could be found?	`FALSE`
`heuristics/fixandinfer/freq` ($-1 \leq$ integer) frequency for calling primal heuristic <fixandinfer> (-1: never, 0: only at depth freqofs)	-1
`heuristics/fixandinfer/freqofs` ($0 \leq$ integer)	0

frequency offset for calling primal heuristic <fixandinfer>

`heuristics/fracdiving/freq` ($-1 \leq$ integer) 10
frequency for calling primal heuristic <fracdiving> (-1: never, 0: only at depth freqofs)

`heuristics/fracdiving/freqofs` ($0 \leq$ integer) 3
frequency offset for calling primal heuristic <fracdiving>

`heuristics/fracdiving/maxlpiterquot` ($0 \leq$ real) 0.05
maximal fraction of diving LP iterations compared to node LP iterations

`heuristics/fracdiving/maxlpiterofs` ($0 \leq$ integer) 1000
additional number of allowed LP iterations

`heuristics/fracdiving/backtrack` (`boolean`) `TRUE`
use one level of backtracking if infeasibility is encountered?

`heuristics/guideddiving/freq` ($-1 \leq$ integer) 10
frequency for calling primal heuristic <guideddiving> (-1: never, 0: only at depth freqofs)

`heuristics/guideddiving/freqofs` ($0 \leq$ integer) 7
frequency offset for calling primal heuristic <guideddiving>

`heuristics/guideddiving/maxlpiterquot` ($0 \leq$ real) 0.05
maximal fraction of diving LP iterations compared to node LP iterations

`heuristics/guideddiving/maxlpiterofs` ($0 \leq$ integer) 1000
additional number of allowed LP iterations

`heuristics/guideddiving/backtrack` (`boolean`) `TRUE`
use one level of backtracking if infeasibility is encountered?

`heuristics/intdiving/freq` ($-1 \leq$ integer) -1
frequency for calling primal heuristic <intdiving> (-1: never, 0: only at depth freqofs)

`heuristics/intdiving/freqofs` ($0 \leq$ integer) 9
frequency offset for calling primal heuristic <intdiving>

`heuristics/intdiving/maxlpiterquot` ($0 \leq$ real) 0.05
maximal fraction of diving LP iterations compared to node LP iterations

`heuristics/intdiving/maxlpiterofs` ($0 \leq$ integer) 1000
additional number of allowed LP iterations

`heuristics/intdiving/backtrack` (`boolean`) `TRUE`
use one level of backtracking if infeasibility is encountered?

`heuristics/intshifting/freq` ($-1 \leq$ integer) 10
frequency for calling primal heuristic <intshifting> (-1: never, 0: only at depth freqofs)

`heuristics/intshifting/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <intshifting>

`heuristics/linesearchdiving/freq` ($-1 \leq$ integer) 10
frequency for calling primal heuristic <linesearchdiving> (-1: never, 0: only at depth freqofs)

`heuristics/linesearchdiving/freqofs` ($0 \leq$ integer) 6
frequency offset for calling primal heuristic <linesearchdiving>

`heuristics/linesearchdiving/maxlpiterquot` ($0 \leq$ real) 0.05
maximal fraction of diving LP iterations compared to node LP iterations

`heuristics/linesearchdiving/maxlpiterofs` ($0 \leq$ integer) 1000
additional number of allowed LP iterations

`heuristics/linesearchdiving/backtrack` (`boolean`) `TRUE`
use one level of backtracking if infeasibility is encountered?

`heuristics/localbranching/freq` ($-1 \leq$ integer) -1

frequency for calling primal heuristic <localbranching> (-1: never, 0: only at depth freqofs)

`heuristics/localbranching/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <localbranching>

`heuristics/localbranching/nodesofs` ($0 \leq$ integer) 1000
number of nodes added to the contingent of the total nodes

`heuristics/localbranching/neighborhoodsize` ($1 \leq$ integer) 18
radius (using Manhattan metric) of the incumbent's neighborhood to be searched

`heuristics/localbranching/nodesquot` ($0 \leq$ real ≤ 1) 0.05
contingent of sub problem nodes in relation to the number of nodes of the original problem

`heuristics/mutation/freq` ($-1 \leq$ integer) -1
frequency for calling primal heuristic <mutation> (-1: never, 0: only at depth freqofs)

`heuristics/mutation/freqofs` ($0 \leq$ integer) 8
frequency offset for calling primal heuristic <mutation>

`heuristics/mutation/nodesofs` ($0 \leq$ integer) 500
number of nodes added to the contingent of the total nodes

`heuristics/mutation/nodesquot` ($0 \leq$ real ≤ 1) 0.1
contingent of sub problem nodes in relation to the number of nodes of the original problem

`heuristics/mutation/minfixingrate` ($10^{-6} \leq$ real ≤ 0.999999) 0.8
percentage of integer variables that have to be fixed

`heuristics/objpscostdiving/freq` ($-1 \leq$ integer) 20
frequency for calling primal heuristic <objpscostdiving> (-1: never, 0: only at depth freqofs)

`heuristics/objpscostdiving/freqofs` ($0 \leq$ integer) 4
frequency offset for calling primal heuristic <objpscostdiving>

`heuristics/objpscostdiving/maxlpiterquot` ($0 \leq$ real ≤ 1) 0.01
maximal fraction of diving LP iterations compared to total iteration number

`heuristics/objpscostdiving/maxlpiterofs` ($0 \leq$ integer) 1000
additional number of allowed LP iterations

`heuristics/octane/freq` ($-1 \leq$ integer) -1
frequency for calling primal heuristic <octane> (-1: never, 0: only at depth freqofs)

`heuristics/octane/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <octane>

`heuristics/oneopt/freq` ($-1 \leq$ integer) 1
frequency for calling primal heuristic <oneopt> (-1: never, 0: only at depth freqofs)

`heuristics/oneopt/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <oneopt>

`heuristics/pscostdiving/freq` ($-1 \leq$ integer) 10
frequency for calling primal heuristic <pscostdiving> (-1: never, 0: only at depth freqofs)

`heuristics/pscostdiving/freqofs` ($0 \leq$ integer) 2
frequency offset for calling primal heuristic <pscostdiving>

`heuristics/pscostdiving/maxlpiterquot` ($0 \leq$ real) 0.05
maximal fraction of diving LP iterations compared to node LP iterations

`heuristics/pscostdiving/maxlpiterofs` ($0 \leq$ integer) 1000
additional number of allowed LP iterations

`heuristics/pscostdiving/backtrack` (`boolean`) `TRUE`
use one level of backtracking if infeasibility is encountered?

`heuristics/rens/freq` ($-1 \leq$ integer) 0 frequency for calling primal heuristic <rens> (-1: never, 0: only at depth freqofs)

heuristics/rens/freqofs ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <rens>

heuristics/rens/minfixingrate ($0 \leq$ real ≤ 1) 0.5
minimum percentage of integer variables that have to be fixable

heuristics/rens/nodesofs ($0 \leq$ integer ≤ -1) 500
number of nodes added to the contingent of the total nodes

heuristics/rens/nodesquot ($0 \leq$ real ≤ 1) 0.1
contingent of sub problem nodes in relation to the number of nodes of the original problem

heuristics/rins/freq ($-1 \leq$ integer) -1
frequency for calling primal heuristic <rins> (-1: never, 0: only at depth freqofs)

heuristics/rins/freqofs ($0 \leq$ integer) 5
frequency offset for calling primal heuristic <rins>

heuristics/rins/nodesofs ($0 \leq$ integer) 500
number of nodes added to the contingent of the total nodes

heuristics/rins/nodesquot ($0 \leq$ real ≤ 1) 0.1
contingent of sub problem nodes in relation to the number of nodes of the original problem

heuristics/rins/minfixingrate ($0 \leq$ real ≤ 1) 0
minimum percentage of integer variables that have to be fixed

heuristics/rootsoldiving/freq ($-1 \leq$ integer) 20
frequency for calling primal heuristic <rootsoldiving> (-1: never, 0: only at depth freqofs)

heuristics/rootsoldiving/freqofs ($0 \leq$ integer) 5
frequency offset for calling primal heuristic <rootsoldiving>

heuristics/rootsoldiving/maxlpiterquot ($0 \leq$ real) 0.01
maximal fraction of diving LP iterations compared to node LP iterations

heuristics/rootsoldiving/maxlpiterofs ($0 \leq$ integer) 1000
additional number of allowed LP iterations

heuristics/rounding/freq ($-1 \leq$ integer) 1
frequency for calling primal heuristic <rounding> (-1: never, 0: only at depth freqofs)

heuristics/rounding/freqofs ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <rounding>

heuristics/shiftandpropagate/freq ($-1 \leq$ integer) 0
frequency for calling primal heuristic <shiftandpropagate> (-1: never, 0: only at depth freqofs)

heuristics/shiftandpropagate/freqofs ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <shiftandpropagate>

heuristics/shifting/freq ($-1 \leq$ integer) 10
frequency for calling primal heuristic <shifting> (-1: never, 0: only at depth freqofs)

heuristics/shifting/freqofs ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <shifting>

heuristics/simplerounding/freq ($-1 \leq$ integer) 1
frequency for calling primal heuristic <simplerounding> (-1: never, 0: only at depth freqofs)

heuristics/simplerounding/freqofs ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <simplerounding>

heuristics/subnlp/freq ($-1 \leq$ integer) 1
frequency for calling primal heuristic <subnlp> (-1: never, 0: only at depth freqofs)

heuristics/subnlp/freqofs ($0 \leq$ integer) 0 frequency offset for calling primal heuristic <subnlp>

`heuristics/subnlp/nlpverblevel` ($0 \leq$ integer) 0
verbosity level of NLP solver

`heuristics/subnlp/nlpiterlimit` ($0 \leq$ integer) 0
iteration limit of NLP solver; 0 to use solver default

`heuristics/subnlp/nlptimelimit` ($0 \leq$ real) 0
time limit of NLP solver; 0 to use solver default

`heuristics/subnlp/nlpsolver` (`string`)
name of an NLP solver to use (empty value means to use solver with highest priority)

`heuristics/subnlp/iteroffset` ($0 \leq$ integer) 500
number of iterations added to the contingent of the total number of iterations

`heuristics/subnlp/iterquotient` ($0 \leq$ real) 0.1
contingent of NLP iterations in relation to the number of nodes in SCIP

`heuristics/subnlp/itermin` ($0 \leq$ integer) 300
contingent of NLP iterations in relation to the number of nodes in SCIP

`heuristics/subnlp/runalways` (`boolean`) `FALSE`
whether to run NLP heuristic always if starting point available (does not use iteroffset,iterquot,itermin)

`heuristics/subnlp/forbidfixings` (`boolean`) `TRUE`
whether to add constraints that forbid specific fixings that turned out to be infeasible

`heuristics/trivial/freq` ($-1 \leq$ integer) 0
frequency for calling primal heuristic <trivial> (-1: never, 0: only at depth freqofs)

`heuristics/trivial/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <trivial>

`heuristics/trysol/freq` ($-1 \leq$ integer) 1
frequency for calling primal heuristic <trysol> (-1: never, 0: only at depth freqofs)

`heuristics/trysol/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <trysol>

`heuristics/twoopt/freq` ($-1 \leq$ integer) -1
frequency for calling primal heuristic <twoopt> (-1: never, 0: only at depth freqofs)

`heuristics/twoopt/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <twoopt>

`heuristics/undercover/freq` ($-1 \leq$ integer) 0
frequency for calling primal heuristic <undercover> (-1: never, 0: only at depth freqofs)

`heuristics/undercover/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <undercover>

`heuristics/undercover/fixingalts` (`string`) `li`
prioritized sequence of fixing values used ('l'p relaxation, 'n'lp relaxation, 'i'ncumbent solution)

`heuristics/undercover/nodesofs` ($0 \leq$ integer ≤ -1) 500
number of nodes added to the contingent of the total nodes

`heuristics/undercover/nodesquot` ($0 \leq$ real ≤ 1) 0.1
contingent of sub problem nodes in relation to the number of nodes of the original problem

`heuristics/undercover/onlyconvexify` (`boolean`) `FALSE`
should we only fix variables in order to obtain a convex problem?

`heuristics/undercover/postnlp` (`boolean`) `TRUE`
should the nlp heuristic be called to polish a feasible solution?

`heuristics/veclendiving/freq` ($-1 \leq$ integer) 10 frequency for calling primal heuristic <veclendiving> (-1: never, 0: only at depth freqofs)

`heuristics/veclendiving/freqofs` ($0 \leq$ integer) 4
frequency offset for calling primal heuristic <veclendiving>

`heuristics/veclendiving/maxlpiterquot` ($0 \leq$ real) 0.05
maximal fraction of diving LP iterations compared to node LP iterations

`heuristics/veclendiving/maxlpiterofs` ($0 \leq$ integer) 1000
additional number of allowed LP iterations

`heuristics/veclendiving/backtrack` (`boolean`) `TRUE`
use one level of backtracking if infeasibility is encountered?

`heuristics/zirounding/freq` ($-1 \leq$ integer) 1
frequency for calling primal heuristic <zirounding> (-1: never, 0: only at depth freqofs)

`heuristics/zirounding/freqofs` ($0 \leq$ integer) 0
frequency offset for calling primal heuristic <zirounding>

Limits

`limits/time` ($0 \leq$ real) 1000
maximal time in seconds to run

`limits/nodes` ($-1 \leq$ integer ≤ -1) -1
maximal number of nodes to process (-1: no limit)

`limits/stallnodes` ($-1 \leq$ integer ≤ -1) -1
solving stops, if the given number of nodes was processed since the last improvement of the primal solution value (-1: no limit)

`limits/memory` ($0 \leq$ real) ∞
maximal memory usage in MB; reported memory usage is lower than real memory usage!

`limits/gap` ($0 \leq$ real) 0.1
solving stops, if the relative gap = —(primalbound - dualbound)/dualbound— is below the given value

`limits/absgap` ($0 \leq$ real) 0
solving stops, if the absolute gap = —primalbound - dualbound— is below the given value

`limits/solutions` ($-1 \leq$ integer) -1
solving stops, if the given number of solutions were found (-1: no limit)

`limits/bestsol` ($-1 \leq$ integer) -1
solving stops, if the given number of solution improvements were found (-1: no limit)

`limits/maxsol` ($1 \leq$ integer) 100
maximal number of solutions to store in the solution storage

`limits/restarts` ($-1 \leq$ integer) -1
solving stops, if the given number of restarts was triggered (-1: no limit)

LP

`lp/solvefreq` ($-1 \leq$ integer) 1
frequency for solving LP at the nodes (-1: never; 0: only root LP)

`lp/solvedepth` ($-1 \leq$ integer) -1
maximal depth for solving LP at the nodes (-1: no depth limit)

`lp/initalgorithm` (`character`) `s`
LP algorithm for solving initial LP relaxations (automatic 's'implex, 'p'rimal simplex, 'd'ual simplex, 'b'arrier, barrier with 'c'rossover)

`lp/resolvealgorithm` (`character`) `s`
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)

`lp/pricing` (`character`) `l`
LP pricing strategy ('l'pi default, 'a'uto, 'f'ull pricing, 'p'artial, 's'teepest edge pricing, 'q'uickstart steepest edge pricing, 'd'evex pricing)

Memory

`memory/savefac` ($0 \leq$ real ≤ 1) 0.8
fraction of maximal memory usage resulting in switch to memory saving mode

Micellaneous

`misc/catchctrlc` (`boolean`) `TRUE`
should the CTRL-C interrupt be caught by SCIP?

`misc/usevartable` (`boolean`) `TRUE`
should a hashtable be used to map from variable names to variables?

`misc/useconstable` (`boolean`) `TRUE`
should a hashtable be used to map from constraint names to constraints?

`misc/usesmalltables` (`boolean`) `FALSE`
should smaller hashtables be used? yields better performance for small problems with about 100 variables

`misc/permutationseed` ($-1 \leq$ integer) -1
seed value for permuting the problem after the problem was transformed (-1: no permutation)

Node Selection

`nodeselection/childsel` (`character`) `h`
child selection rule ('d'own, 'u'p, 'p'seudo costs, 'i'nference, 'l'p value, 'r'oot LP value difference, 'h'brid inference/root LP value difference)

`nodeselection/bfs/stdpriority` ($-536870912 \leq$ integer ≤ 536870911) 100000
priority of node selection rule <bfs> in standard mode

`nodeselection/dfs/stdpriority` ($-536870912 \leq$ integer ≤ 536870911) 0
priority of node selection rule <dfs> in standard mode

`nodeselection/estimate/stdpriority` ($-536870912 \leq$ integer ≤ 536870911) 200000
priority of node selection rule <estimate> in standard mode

`nodeselection/estimate/bestnodefreq` ($0 \leq$ integer) 10
frequency at which the best node instead of the best estimate is selected (0: never)

`nodeselection/hybridestim/stdpriority` ($-536870912 \leq$ integer ≤ 536870911) 50000
priority of node selection rule <hybridestim> in standard mode

`nodeselection/hybridestim/bestnodefreq` ($0 \leq$ integer) 1000
frequency at which the best node instead of the hybrid best estimate / best bound is selected (0: never)

`nodeselection/restartdfs/stdpriority` ($-536870912 \leq$ integer ≤ 536870911) 10000
priority of node selection rule <restartdfs> in standard mode

`nodeselection/restartdfs/selectbestfreq` ($0 \leq$ integer) 0
frequency for selecting the best node instead of the deepest one

Tolerances

`numerics/epsilon` ($10^{-20} \leq$ real ≤ 0.001) 10^{-9}
absolute values smaller than this are considered zero

`numerics/sumepsilon` ($10^{-17} \leq$ real ≤ 0.001) 10^{-6}
absolute values of sums smaller than this are considered zero

`numerics/feastol` ($10^{-17} \leq$ real ≤ 0.001) 10^{-6}
feasibility tolerance for constraints

`numerics/dualfeastol` ($10^{-17} \leq$ real ≤ 0.001) 10^{-9}
feasibility tolerance for reduced costs in LP solution

Presolving

`presolving/maxrounds` $(-1 \leq \text{integer})$ -1
maximal number of presolving rounds (-1: unlimited, 0: off)

`presolving/maxrestarts` $(-1 \leq \text{integer})$ -1
maximal number of restarts (-1: unlimited)

`presolving/boundshift/maxrounds` $(-1 \leq \text{integer})$ 0
maximal number of presolving rounds the presolver participates in (-1: no limit)

`presolving/dualfix/maxrounds` $(-1 \leq \text{integer})$ -1
maximal number of presolving rounds the presolver participates in (-1: no limit)

`presolving/implics/maxrounds` $(-1 \leq \text{integer})$ -1
maximal number of presolving rounds the presolver participates in (-1: no limit)

`presolving/inttobinary/maxrounds` $(-1 \leq \text{integer})$ -1
maximal number of presolving rounds the presolver participates in (-1: no limit)

`presolving/probing/maxrounds` $(-1 \leq \text{integer})$ -1
maximal number of presolving rounds the presolver participates in (-1: no limit)

`presolving/probing/maxruns` $(-1 \leq \text{integer})$ 1
maximal number of runs, probing participates in (-1: no limit)

`presolving/trivial/maxrounds` $(-1 \leq \text{integer})$ -1
maximal number of presolving rounds the presolver participates in (-1: no limit)

Domain Propagation

`propagating/maxrounds` $(-1 \leq \text{integer})$ 100
maximal number of propagation rounds per node (-1: unlimited)

`propagating/maxroundsroot` $(-1 \leq \text{integer})$ 1000
maximal number of propagation rounds in the root node (-1: unlimited)

`propagating/abortoncutoff` (`boolean`) `TRUE`
should propagation be aborted immediately? setting this to FALSE could help conflict analysis to produce more conflict constraints

`propagating/pseudoobj/freq` $(-1 \leq \text{integer})$ 1
frequency for calling propagator <pseudoobj> (-1: never, 0: only in root node)

`propagating/rootredcost/freq` $(-1 \leq \text{integer})$ 1
frequency for calling propagator <rootredcost> (-1: never, 0: only in root node)

`propagating/vbounds/freq` $(-1 \leq \text{integer})$ 1
frequency for calling propagator <vbounds> (-1: never, 0: only in root node)

Separation

`separating/maxbounddist` $(0 \leq \text{real} \leq 1)$ 1
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)

`separating/minefficacy` $(0 \leq \text{real})$ 0.05
minimal efficacy for a cut to enter the LP

`separating/minefficacyroot` $(0 \leq \text{real})$ 0.01
minimal efficacy for a cut to enter the LP in the root node

`separating/minortho` $(0 \leq \text{real} \leq 1)$ 0.5
minimal orthogonality for a cut to enter the LP

`separating/minorthoroot` $(0 \leq \text{real} \leq 1)$ 0.5
minimal orthogonality for a cut to enter the LP in the root node

`separating/maxrounds` $(-1 \leq \text{integer})$ 5 maximal number of separation rounds per node (-1: unlimited)

`separating/maxroundsroot` ($-1 \leq$ integer) -1
maximal number of separation rounds in the root node (-1: unlimited)

`separating/maxstallrounds` ($-1 \leq$ integer) 5
maximal number of consecutive separation rounds without objective or integrality improvement (-1: no additional restriction)

`separating/maxcuts` ($0 \leq$ integer) 100
maximal number of cuts separated per separation round (0: disable local separation)

`separating/maxcutsroot` ($0 \leq$ integer) 2000
maximal number of separated cuts at the root node (0: disable root node separation)

`separating/poolfreq` ($-1 \leq$ integer) 0
separation frequency for the global cut pool (-1: disable global cut pool, 0: only separate pool at the root)

`separating/clique/freq` ($-1 \leq$ integer) 0
frequency for calling separator <clique> (-1: never, 0: only in root node)

`separating/clique/maxsepacuts` ($-1 \leq$ integer) 10
maximal number of clique cuts separated per separation round (-1: no limit)

`separating/cgmip/freq` ($-1 \leq$ integer) -1
frequency for calling separator <cgmip> (-1: never, 0: only in root node)

`separating/cgmip/maxrounds` ($-1 \leq$ integer) 0
maximal number of cgmip separation rounds per node (-1: unlimited)

`separating/cgmip/maxroundsroot` ($-1 \leq$ integer) 50
maximal number of cgmip separation rounds in the root node (-1: unlimited)

`separating/cgmip/dynamiccuts` (`boolean`) `TRUE`
should generated cuts be removed from the LP if they are no longer tight?

`separating/cgmip/nodelimit` ($-1 \leq$ integer ≤ -1) 10000
node limit for sub-MIP (-1: unlimited)

`separating/cgmip/usecmir` (`boolean`) `TRUE`
use CMIR-generator (otherwise add cut directly)?

`separating/cgmip/cmirownbounds` (`boolean`) `FALSE`
tell CMIR-generator which bounds to used in rounding?

`separating/cgmip/allowlocal` (`boolean`) `FALSE`
allow to generate local cuts?

`separating/cgmip/onlyintvars` (`boolean`) `FALSE`
generate cuts for problems with only integer variables?

`separating/cgmip/onlyactiverows` (`boolean`) `TRUE`
use only active rows to generate cuts?

`separating/cgmip/usecutpool` (`boolean`) `TRUE`
use cutpool to store CG-cuts even if the are not efficient?

`separating/cgmip/primalseparation` (`boolean`) `TRUE`
only separate cuts that are tight for the best feasible solution?

`separating/cgmip/onlyrankone` (`boolean`) `FALSE`
whether only rank 1 inequalities should be separated

`separating/cgmip/earlyterm` (`boolean`) `TRUE`
terminate separation if a violated (but possibly sub-optimal) cut has been found?

`separating/cgmip/addviolationcons` (`boolean`) `TRUE`
add constraint to subscip that only allows violated cuts?

`separating/cgmip/addviolconshdlr` (boolean) add constraint handler to filter out violated cuts?	`FALSE`
`separating/cgmip/conshdlrusenorm` (boolean) should the violation constraint handler use the norm of a cut to check for feasibility?	`TRUE`
`separating/cgmip/objlone` (boolean) should the objective of the sub-MIP minimize the l1-norm of the multipliers?	`FALSE`
`separating/cmir/freq` ($-1 \leq$ integer) frequency for calling separator <cmir> (-1: never, 0: only in root node)	0
`separating/cmir/maxrounds` ($-1 \leq$ integer) maximal number of cmir separation rounds per node (-1: unlimited)	3
`separating/cmir/maxroundsroot` ($-1 \leq$ integer) maximal number of cmir separation rounds in the root node (-1: unlimited)	10
`separating/cmir/maxsepacuts` ($0 \leq$ integer) maximal number of cmir cuts separated per separation round	100
`separating/cmir/maxsepacutsroot` ($0 \leq$ integer) maximal number of cmir cuts separated per separation round in the root node	500
`separating/cmir/dynamiccuts` (boolean) should generated cuts be removed from the LP if they are no longer tight?	`TRUE`
`separating/flowcover/freq` ($-1 \leq$ integer) frequency for calling separator <flowcover> (-1: never, 0: only in root node)	0
`separating/flowcover/maxrounds` ($-1 \leq$ integer) maximal number of separation rounds per node (-1: unlimited)	5
`separating/flowcover/maxroundsroot` ($-1 \leq$ integer) maximal number of separation rounds in the root node (-1: unlimited)	10
`separating/flowcover/maxsepacuts` ($0 \leq$ integer) maximal number of flow cover cuts separated per separation round	100
`separating/flowcover/maxsepacutsroot` ($0 \leq$ integer) maximal number of flow cover cuts separated per separation round in the root	200
`separating/flowcover/dynamiccuts` (boolean) should generated cuts be removed from the LP if they are no longer tight?	`TRUE`
`separating/gomory/freq` ($-1 \leq$ integer) frequency for calling separator <gomory> (-1: never, 0: only in root node)	0
`separating/gomory/maxrounds` ($-1 \leq$ integer) maximal number of gomory separation rounds per node (-1: unlimited)	5
`separating/gomory/maxroundsroot` ($-1 \leq$ integer) maximal number of gomory separation rounds in the root node (-1: unlimited)	-1
`separating/gomory/maxsepacuts` ($0 \leq$ integer) maximal number of gomory cuts separated per separation round	50
`separating/gomory/maxsepacutsroot` ($0 \leq$ integer) maximal number of gomory cuts separated per separation round in the root node	500
`separating/gomory/dynamiccuts` (boolean) should generated cuts be removed from the LP if they are no longer tight?	`TRUE`
`separating/impliedbounds/freq` ($-1 \leq$ integer) frequency for calling separator <impliedbounds> (-1: never, 0: only in root node)	0
`separating/intobj/freq` ($-1 \leq$ integer) frequency for calling separator <intobj> (-1: never, 0: only in root node)	-1

`separating/mcf/freq` ($-1 \leq$ integer) 0
frequency for calling separator <mcf> (-1: never, 0: only in root node)

`separating/mcf/dynamiccuts` (`boolean`) `TRUE`
should generated cuts be removed from the LP if they are no longer tight?

`separating/mcf/maxsepacuts` ($-1 \leq$ integer) 100
maximal number of mcf cuts separated per separation round

`separating/mcf/maxsepacutsroot` ($-1 \leq$ integer) 200
maximal number of mcf cuts separated per separation round in the root node – default separation

`separating/oddcycle/freq` ($-1 \leq$ integer) -1
frequency for calling separator <oddcycle> (-1: never, 0: only in root node)

`separating/oddcycle/useclassical` (`boolean`) `TRUE`
should classical search method by Groetschel, Lovasz, Schrijver be used? Otherwise use levelgraph method by Hoffman, Padberg.

`separating/oddcycle/liftoddcycles` (`boolean`) `FALSE`
should odd cycle cuts be lifted?

`separating/oddcycle/maxsepacuts` ($0 \leq$ integer) 5000
maximal number of oddcycle cuts separated per separation round

`separating/oddcycle/maxsepacutsroot` ($0 \leq$ integer) 5000
maximal number of oddcycle cuts separated per separation round in the root node

`separating/oddcycle/maxrounds` ($-1 \leq$ integer) 10
maximal number of oddcycle separation rounds per node (-1: unlimited)

`separating/oddcycle/maxroundsroot` ($-1 \leq$ integer) 10
maximal number of oddcycle separation rounds in the root node (-1: unlimited)

`separating/rapidlearning/freq` ($-1 \leq$ integer) -1
frequency for calling separator <rapidlearning> (-1: never, 0: only in root node)

`separating/redcost/freq` ($-1 \leq$ integer) 1
frequency for calling separator <redcost> (-1: never, 0: only in root node)

`separating/redcost/continuous` (`boolean`) `FALSE`
should reduced cost fixing be also applied to continuous variables?

`separating/strongcg/freq` ($-1 \leq$ integer) 0
frequency for calling separator <strongcg> (-1: never, 0: only in root node)

`separating/strongcg/maxrounds` ($-1 \leq$ integer) 5
maximal number of strong CG separation rounds per node (-1: unlimited)

`separating/strongcg/maxroundsroot` ($-1 \leq$ integer) 20
maximal number of strong CG separation rounds in the root node (-1: unlimited)

`separating/strongcg/maxsepacuts` ($0 \leq$ integer) 50
maximal number of strong CG cuts separated per separation round

`separating/strongcg/maxsepacutsroot` ($0 \leq$ integer) 500
maximal number of strong CG cuts separated per separation round in the root node

`separating/strongcg/dynamiccuts` (`boolean`) `TRUE`
should generated cuts be removed from the LP if they are no longer tight?

`separating/zerohalf/freq` ($-1 \leq$ integer) -1
frequency for calling separator <zerohalf> (-1: never, 0: only in root node)

`separating/zerohalf/maxrounds` ($-1 \leq$ integer) 5
maximal number of zerohalf separation rounds per node (-1: unlimited)

`separating/zerohalf/maxroundsroot` ($-1 \leq$ integer) 10
maximal number of zerohalf separation rounds in the root node (-1: unlimited)

`separating/zerohalf/maxsepacuts` ($0 \leq$ integer) 50
maximal number of 0,1/2-cuts separated per separation round

`separating/zerohalf/maxsepacutsroot` ($0 \leq$ integer) 500
maximal number of 0,1/2-cuts separated per separation round in the root node

`separating/zerohalf/dynamiccuts` (`boolean`) `TRUE`
should generated cuts be removed from the LP if they are no longer tight?

`separating/zerohalf/preprocessing/decomposeproblem` (`boolean`) `FALSE`
should problem be decomposed into subproblems (if possible) before applying preprocessing?

`separating/zerohalf/preprocessing/delta` ($0 \leq$ real ≤ 1) 0.5
value of delta parameter used in preprocessing method 'd'

`separating/zerohalf/preprocessing/ppmethods` (`string`) `CXGXIM`
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

`separating/zerohalf/separating/forcecutstolp` (`boolean`) `FALSE`
should the cuts be forced to enter the LP?

`separating/zerohalf/separating/forcecutstosepastore` (`boolean`) `FALSE`
should the cuts be forced to enter SCIP's sepastore?

`separating/zerohalf/separating/minviolation` ($0.001 \leq$ real ≤ 0.5) 0.3
minimal violation of a 0,1/2-cut to be separated

`separating/zerohalf/separating/sepamethods` (`string`) 2g
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

`separating/zerohalf/separating/auxip/settingsfile` (`string`) `-`
optional settings file of the auxiliary IP (-: none)

`separating/zerohalf/separating/auxip/sollimit` ($-1 \leq$ integer) -1
limits/solutions setting of the auxiliary IP

`separating/zerohalf/separating/auxip/penaltyfactor` ($0 \leq$ real ≤ 1) 0.001
penalty factor used with objective function 'p' of auxiliary IP

`separating/zerohalf/separating/auxip/useallsols` (`boolean`) `TRUE`
should all (proper) solutions of the auxiliary IP be used to generate cuts instead of using only the best?

`separating/zerohalf/separating/auxip/objective` (`character`) `v`
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

Timing

`timing/clocktype` ($1 \leq$ integer ≤ 2) 1
default clock type (1: CPU user seconds, 2: wall clock time)

`timing/enabled` (`boolean`) `TRUE`
is timing enabled?

SCIP References

[1] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[2] Tobias Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computations*, 1(1):1–41, 2009.

[3] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: A new approach to integrate CP and MIP. In L. Perron and M.A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008*, volume 5015 of *LNCS*, pages 6–20. Springer, 2008.

[4] Timo Berthold. Primal heuristics for mixed integer programs. Diploma thesis, Technische Universität Berlin, 2006.

[5] Timo Berthold, Stefan Heinz, and Stefan Vigerske. Extending a CIP framework to solve MIQCPs. In Jon Lee and Sven Leyffer, editors, *Mixed-integer nonlinear optimization: Algorithmic advances and applications*, IMA volumes in Mathematics and its Applications. Springer, 2009. to appear.

[6] Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. `http://projects.coin-or.org/Ipopt`.

[7] Kati Wolter. Implementation of cutting plane separators for mixed integer programs. Diploma thesis, Technische Universität Berlin, 2006.

[8] Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. `http://www.zib.de/Publications/abstracts/TR-96-09`, `http://soplex.zib.de`.

SNOPT

Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA

Walter Murray, Michael A. Saunders; Department of EESOR, Stanford University, Stanford, CA

Arne Drud; ARKI Consulting and Development, Bagsvaerd, Denmark

Contents

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

$$\boxed{\begin{array}{ll} \text{NP} & \displaystyle\operatorname*{minimize}_x \text{ or } \operatorname*{maximize}_x f_0(x) \\ & \begin{array}{rl} & f(x) \sim b_1 \\ \text{subject to} & A_L x \sim b_2 \\ & l \le x \le u, \end{array} \end{array}} \qquad (1.1)$$

where $x \in \Re^n$, $f_0(x)$ 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, $\sim$ 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 `-INF` or `+INF` to indicate $l_j = -\infty$ or $u_j = +\infty$ for appropriate j. Free variables have both bounds infinite and fixed variables have $l_j = u_j$.

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 [13] as in MINOS [16].

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 [6], 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 [9, 11].

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:

```
option NLP=SNOPT;
```

If the model contains non-smooth functions like abs(x), or max(x, y) you can try to get it solved by SNOPT using

```
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:

```
> 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 by Gill, Murray and Saunders[7].

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:

```
obj.. z =e= sum(i, sqr[r(i)]);

model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form (1.1) 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

```
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

```
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

$$\text{minimize}\, f(x) = (1+x)^2.$$

These additional reformulations can be turned off by using the statement `option reform = 0;` (see §4.1).

2.2 Constraints and slack variables

Problem (1.1) contains n variables in x. Let m be the number of components of $f(x)$ and $A_L x$ 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 $s = (s_1, s_2, ..., s_m)^T$. For example, the linear constraint $5 \le 2x_1 + 3x_2 \le +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \le s_1 \le +\infty$. Problem (1.1) can be written in the equivalent form

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & f_0(x) \\ \text{subject to} & \begin{pmatrix} f(x) \\ A_L x \end{pmatrix} - s = 0, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} \le u. \end{array}$$

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 $A_L x - s_L = 0$, 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 $A_L x - s_L = 0$ 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

$$\begin{array}{rccl} f'(x_k)x & -s_N & & = -f(x_k) + f'(x_k)x_k, \\ A_L x & & -s_L & = \ 0, \end{array}$$

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

$$A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\boxed{\begin{array}{lll} \text{QP}_k & \underset{x,s}{\text{minimize}} & q(x, x_k) = g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T H_k (x - x_k) \\ & \text{subject to} & Ax - s = b, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} \le u, \end{array}} \tag{2.2}$$

where $q(x, x_k)$ is a quadratic approximation to a modified Lagrangian function [6]. 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[8] 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

$$\begin{pmatrix} B & S & N \\ & & I \end{pmatrix} \begin{pmatrix} x_B \\ x_S \\ x_N \end{pmatrix} = \begin{pmatrix} b \\ x_N \end{pmatrix},$$

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:

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix} \tag{2.3}$$

where P permutes the columns of $(A\ -I)$ into the order $(B\ S\ N)$. Minimizing $q(x, x_k)$ with respect to p_S now involves a quadratic function of p_S:

$$g^T Zp_S + \frac{1}{2} p_S^T Z^T HZp_S, \tag{2.4}$$

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 $Z^T HZ$ (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

$$Z^T HZp_S = -Z^T g. \tag{2.5}$$

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[13] 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

$$B^T \pi = g_B, \ \ Z^T g = g_S - S^T \pi,$$

where π 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.

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):

$$d = g - \begin{pmatrix} A^T \\ -I \end{pmatrix} \pi, \quad \text{so that } d_S = Z^T g.$$

At a feasible point, the reduced gradients for the slacks s are the dual variables π.

The optimality conditions for subproblem QP$_k$ (2.2) may be written in terms of d. The current point is optimal if $d_j \ge 0$ for all nonbasic variables at their lower bounds, $d_j \le 0$ 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 $(\widehat{x}_k, \widehat{s}_k, \widehat{\pi}_k)$ 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 $(x, s) + p$ 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 testing reduced gradients d_N is known as *pricing* (a term introduced in the context of the simplex method for linear programming). Pricing the jth variable means computing $d_j = g_j - a_j^T \pi$, where a_j is the jth 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 (2.5) 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, $n_S \le 1000$). 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

$$\mathcal{M}(x, s, \pi) = f(x) - \pi^T\big(F(x) - s_N\big) + \tfrac{1}{2}\big(F(x) - s_N\big)^T D\big(F(x) - s_N\big), \tag{2.6}$$

where D is a diagonal matrix of penalty parameters. If (x_k, s_k, π_k) denotes the current solution estimate and $(\widehat{x}_k, \widehat{s}_k, \widehat{\pi}_k)$ denotes the optimal QP solution, the linesearch determines a step α_k $(0 < \alpha_k \le 1)$ such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \widehat{x}_k - x_k \\ \widehat{s}_k - s_k \\ \widehat{\pi}_k - \pi_k \end{pmatrix} \tag{2.7}$$

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 $\mathcal{M}$ [11]. 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 [9, 3].

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

$$\boxed{\begin{array}{lll} \text{FLP} \qquad\qquad & \underset{x,v,w}{\text{minimize}} & e^T(v + w) \\ & \text{subject to} & \ell \le \begin{pmatrix} x \\ A_L x - v + w \end{pmatrix} \le u, \; v \ge 0, \; w \ge 0, \end{array}}$$

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 ℓ_1 norm of the infeasibilities.)

If the linear constraints are infeasible ($v \ne 0$ or $w \ne 0$), 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 (1.1) as given, using search directions obtained from a sequence of quadratic programming subproblems (2.2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become large), SNOPT enters "elastic" mode and solves the problem

$$\text{NP}(\gamma) \qquad \underset{x,v,w}{\text{minimize}} \quad f_0(x) + \gamma e^T(v+w)$$
$$\text{subject to} \quad \ell \le \begin{pmatrix} x \\ f(x) - v + w \\ A_L x \end{pmatrix} \le u, \; v \ge 0, \; w \ge 0,$$

where γ is a nonnegative parameter (the *elastic weight*), and $f(x) + \gamma e^T(v+w)$ is called a *composite objective.* If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar ℓ_1 formulation of NP is fundamental to the Sℓ_1QP algorithm of Fletcher [4]. See also Conn [1].

The initial value of γ 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):

$$\text{Example} \qquad \underset{r,a,b}{\text{minimize}} \quad r$$
$$\text{subject to} \quad (x_i - a)^2 + (y_i - b)^2 \le r^2, \quad r \ge 0.$$

This problem can be modeled in GAMS as follows.

```
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

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

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:

```
 Model exxon /all/;
 ...
 Solve exxon maximizing profit using lp;
 Solve exxon maximizing profit using lp;
```

we observe the following iteration counts:

```
               S O L V E      S U M M A R Y

     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

.....

               S O L V E      S U M M A R Y

     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 `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 `bratio` option (see §4.1).

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 GAMS options

The following GAMS options affect the behavior of SNOPT.

NLP

This option selects the NLP solver. Example: `option NLP=SNOPT;`. See also §1.2.

DNLP

Selects the DNLP solver for models with discontinuous or non-differentiable functions. Example: `option DNLP=SNOPT;`. See also §1.2.

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=SNOPT;`. See also §1.2.

iterlim

Sets the (minor) iteration limit. Example: `option iterlim=50000;`. The default is 10000. SNOPT 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. SNOPT will stop as soon as more than *reslim* seconds have elapsed since SNOPT 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 $x < 0$ (x^y is evaluated as $\exp(y \log x)$). When such a situation occurs the number of domain errors is increased by one, and SNOPT 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 $\sqrt{x}, x < 0$ a zero is passed back) and SNOPT is asked to continue. In many cases SNOPT 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 SNOPT 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 SNOPT will be added to the listing file. Example: `option sysout=on;`. Default: sysout = off.

work

The `work` option sets the amount of memory SNOPT 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 SNOPT 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 §2.1 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.

4.2 Model suffices

A model identifier in GAMS can have several suffices to which you can assign values. A small GAMS fragment illustrates this:

```
model m /all/;
m.iterlim = 3000;
solve m minimizing z using nlp;
```

Options set by assigning to the suffixed model identifier override the global options. For example,

```
model m /all/;
m.iterlim = 3000;
option iterlim = 100;
solve m minimizing z using nlp;
```

will use an iteration limit of 3000 for this solve.

m.iterlim
Sets the iteration limit. Overrides the global iteration limit. Example: `m.iterlim=50000;` The default is 10000. See also §4.1.

m.reslim
Sets the resource or time limit. Overrides the global resource limit. Example: `m.reslim=600;` The default is 1000 seconds. See also §4.1.

m.bratio
Sets the basis acceptance test parameter. Overrides the global setting. Example: `m.bratio=1.0;` The default is 0.25. See also §4.1.

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 SNOPT will see are scaled by a factor $1/\mathit{variable\_scale}$, so the modeler should use scale factors that represent the order of magnitude of the variable. In that case SNOPT will see variables that are scaled around 1.0. Similarly equation scales can be assigned to equations, which are scaled by a factor $1/\mathit{equation\_scale}$. 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 SNOPT option `scale option`.

m.optfile
Sets whether or not to use a solver option file. Solver specific SNOPT options are specified in a file called `snopt.opt`, see §4.3. To tell SNOPT to use this file, add the statement: `m.optfile=1;`. The default is not to use an option file.

m.workspace
The workspace option sets the amount of memory that SNOPT 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 SNOPT 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;`.

m.workfactor
This increased the available work space for SNOPT by a factor. E.g. `m.workfactor = 2;` will double the available memory.

4.3 SNOPT options

SNOPT options are specified in a file called `snopt.opt` which should reside in the working directory (this is the project directory when running models from the IDE). To tell SNOPT to read this file, use the statement `m.optfile = 1;` in the model (see §4.2).

An example of a valid `snopt.opt` is:

```
Hessian full memory
Hessian frequency 20
```

Users familiar with the SNOPT distribution from Stanford University will notice that the `begin` and `end` keywords are missing. These markers are optional in GAMS/SNOPT and will be ignored. Therefore, the following option file is also accepted:

```
begin
Hessian full memory
Hessian frequency 20
end
```

All options are case-insensitive. A line is a comment line if it starts with an asterisk, `*`, in column one.

Here follows a description of all SNOPT options that are possibly useful in a GAMS environment:

Check frequency i

Every ith 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: `check frequency 60`.

Cold Start

Requests that the CRASH procedure be used to choose an initial basis.

Crash option i

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the constraint matrix $(\,A \quad -I\,)$. 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.

`Crash option 0`: The initial basis contains only slack variables: $B = I$.

`Crash option 1`: CRASH is called once, looking for a triangular basis in all rows and columns of the matrix A.

`Crash option 2`: 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).

`Crash option 3`: 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.

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: `Crash option 3` for linearly constrained problems and `Crash option 0;` for problems with nonlinear constraints.

Crash tolerance r

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 $|a_{ij}| \le a_{\max} \times r$. (To be meaningful, r should be in the range $0 \le r < 1$.)
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 m columns for the initial basis, we would specify `Crash tolerance` r for some value of $r > 0.5$.
Default: `Crash tolerance 0.1`

Derivative level i

The keyword `Derivative level` specifies which nonlinear function gradients are known analytically and will be supplied to SNOPT.

`Derivative level 3:` All objective and constraint gradients are known.

`Derivative level 2:` All constraint gradients are known, but some or all components of the objective gradient are unknown.

`Derivative level 1:` The objective gradient is known, but some or all of the constraint gradients are unknown.

`Derivative level 0:` Some components of the objective gradient are unknown and some of the constraint gradients are unknown.

The value $i = 3$ should be used whenever possible. It is the most reliable and will usually be the most efficient.

Derivative linesearch
Nonderivative linesearch

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 α_k. 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.

Difference interval

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.

Elastic weight ω

This parameter denoted by ω determines the initial weight γ associated with problem NP(γ).

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, γ is increased (subject to a maximum allowable value) at any point that is optimal for problem NP(γ), but not feasible for NP. After the rth increase, $\gamma = \omega 10^r(1 + \|g(x_{k1})\|_2)$, where x_{k1} is the iterate at which γ was first needed.
Default: `Elastic weight 10000.0`

Expand frequency i

This option is part of the EXPAND anti-cycling procedure [10] 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 δ. Over a period of i iterations, the tolerance actually used by SNOPT increases from 0.5δ to δ (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: `Expand frequency 10000`

Factorization frequency k

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.

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 ϵ_R

The *relative function precision* ϵ_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 ϵ_R would be `1.0e-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, ϵ_R should be the absolute precision. For example, if $f(x) =$ `1.23456789e-4` at some point and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_R would be `1.0e-10`.)

- The default value of ϵ_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.

Hessian Full memory

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

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 frequency *i*

If `Hessian Full` is selected and i 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).

Hessian updates *i*

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).

Infinite bound *r*

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.

Default: `Infinite bound 1.0e20`

Iterations limit *k*

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.

Default: specified by GAMS.

Linesearch tolerance *t*

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`.

Log frequency k
See `Print frequency`.
Default: `Log frequency 100`

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 μ 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

$$\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix},$$

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`.

LU partial pivoting
LU rook pivoting
LU complete pivoting
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.)

Default: `LU partial pivoting`.

LU density tolerance r_1
LU singularity tolerance r_2
The density tolerance r_1 is used during LUSOLs 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.

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 $|U_{jj}| \le r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart)

Default: `LU density tolerance 0.6` and `LU singularity tolerance 3.2e-11` for most machines. This value corresponds to $\epsilon^{2/3}$, where ϵ is the relative machine precision.

Major feasibility tolerance ϵ_r

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 `rowerr` be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$\texttt{rowerr} = \max_i \ \texttt{viol}_i/\|x\| \ \le \ \epsilon_r, \tag{4.8}$$

where $\texttt{viol}_i$ is the violation of the ith nonlinear constraint ($i = 1$: `nnCon`, `nnCon` being the number of nonlinear constraints).
In the GAMS/SNOPT iteration log, `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: `Major feasibility tolerance 1.0e-6`.

Major iterations limit k

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` $\max\{1000, m\}$.

Major optimality tolerance ϵ_d

This specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution (x, s, π) such that

$$\texttt{maxComp} = \max_j \ \texttt{Comp}_j/\|\pi\| \ \le \ \epsilon_d, \tag{4.9}$$

where $\texttt{Comp}_j$ is an estimate of the complementarity slackness for variable j ($j = 1 : n + m$). The values $\texttt{Comp}_j$ are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where g_j is the jth component of the objective gradient, a_j is the associated column of the constraint matrix $(\,A \quad -I\,)$, and π is the set of QP dual variables):

$$\texttt{Comp}_j = \begin{cases} d_j \min\{x_j - l_j, 1\} & \text{if } d_j \ge 0; \\ -d_j \min\{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases}$$

In the GAMS/SNOPT iteration log, `maxComp` appears as the quantity labeled "`Optimal`".
Default: `Major optimality tolerance 1.0e-6`.

Major print level p

This controls the amount of output to the GAMS listing file each major iteration. This output is only visible if the `sysout` option is turned on (see §4.1). `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

```
Major print level JFDXbs
```

where each letter stands for a digit that is either `0` or `1` as follows:

- `s` 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` ≥ 1).

- `b` BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if `JFDXbs` ≥ 10).
- `X` x_k, the nonlinear variables involved in the objective function or the constraints.
- `D` π_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: `Major print level 00001`

Major step limit r

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.

I A linesearch determines a step α over the range $0 < \alpha \leq \beta$, where β 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 $\alpha_1 = \min(1, \beta)$.

II 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 $\bar{\beta} = r(1 + \|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.

III 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.

IV In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Default: `Major step limit 2.0`.

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: `Minor iterations limit` 500

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 `sInf` be the corresponding sum of infeasibilities. If `sInf` 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 $t = 1.0\text{e-}6$, 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 `Scale option` ≥ 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: `Minor feasilibility tolerance 1.0e-6`.

Minor print level k
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 §4.1). The value of k has the following effect:

0	No minor iteration output except error messages.
≥ 1	A single line of output each minor iteration (controlled by `Print frequency`).
≥ 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: `Minor print level 1`.

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 $(\,A \quad -I\,)$ 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.

Pivot tolerance r 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 t 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 $\epsilon^{2/3}$ where ϵ is the machine precision.

Print frequency k

When `sysout` is turned on (see §4.1) and `Minor print level` ≥ 1, a line of the QP iteration log will be printed on the listing file every kth minor iteration.
Default: `Print frequency 1`.

Proximal point method i

$i = 1$ 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: `Proximal point method 1`.

QPSolver Cholesky
QPSolver CG
QPSolver QN

This specifies the method used to solve system (2.5) for the search directions in phase 2 of the QP subproblem.

`QPSolver Cholesky` holds the full Cholesky factor R of the reduced Hessian $Z^T HZ$. 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`.

`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.

`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.

- 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: `QPSolver Cholesky`.

Reduced Hessian dimension i

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 $R^T R = Z^T HZ$). The value of i affects when `QPSolver CG` is activated.
Default: `Reduced Hessian dimension`$= \min\{2000, n_1 + 1\}$

Scale option i
Scale tolerance t
Scale Print

Three scale options are available as follows:

`Scale option 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).

`Scale option 1`: 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.

`Scale option 2`: All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of $(A \quad -I)$ 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 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:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \qquad (a_{ij} \neq 0).$$

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.

`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 §4.1).

Defaults: `Scale option 2` for linear models and `Scale option 1` for NLP's, `Scale tolerance 0.9`.

Solution Yes

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 §4.1).
Default: turned off.

Start Objective Check at Column k
Start Constraint Check at Column k
Stop Objective Check at Column l
Stop Constraint Check at Column l

If `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.

Summary frequency k

If `Minor print level`> 0, a line of the QP iteration log is output every kth minor iteration.

Superbasics limit i

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 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 $n_1 + 1$, 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.

Suppress parameters

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 yes
System information no

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.

Unbounded objective value $f_{\max}$
Unbounded step size $\alpha_{\max}$

These parameters are 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 α varies. if $|f_0|$ exceeds $f_{\max}$ or α exceeds $\alpha_{\max}$, 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.

Defaults: `Unbounded objective value 1.0e+15` and `Unbounded step size 1.0e+18`.

Verify level l

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.

- `verify level 0`: Only a "cheap" test will be performed.
- `verify level 1`: Individual gradients will be checked (with a more reliable test).
- `verify level 2`: Individual columns of the problem Jacobian will be checked.
- `verify level 3`: Options 2 and 1 will both occur (in that order).
- `verify level -1`: Derivative checking is disabled.

This option has limited use in a GAMS environment.

Violation limit τ

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

$$v_i(x_{k+1}) \le \tau \max\{1, v_i(x_0)\},$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the ith 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 τ. 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 τ may be any large positive value.
Default: `Violation limit 10`.

Warm start

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.gms Start 10/31/06 16:34:47
GAMS Rev 146  Copyright (C) 1987-2006 GAMS Development. All rights reserved
Licensee: Erwin Kalvelagen                              G060302/0001CE-WIN
          GAMS Development Corporation                              DC4572
--- Starting compilation
--- chem.gms(49) 3 Mb
--- Starting execution
--- chem.gms(45) 4 Mb
--- Generating NLP model mixer
--- chem.gms(49) 6 Mb
---   5 rows  12 columns  37 non-zeroes
---   193 nl-code  11 nl-non-zeroes
--- chem.gms(49) 4 Mb
--- Executing SNOPT

SNOPT-Link    BETA  1Nov06 WIN.SN.SN 22.3 043.058.041.VIS SNOPT 7.2-4

    GAMS/SNOPT, Large Scale Nonlinear SQP Solver
    S N O P T  7.2-4 (June 2006)
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.21 Mb

  Reading Rows...
  Reading Columns...
  Reading Instructions...

 Major Minor   Step   nObj   Objective   Optimal   nS PD
     0     7 0.0E+00     1  3.292476E+01 1.8E-01    5 TF n r
     1     2 4.1E+00     3  4.131132E+01 1.6E-01    6 TF n r
     2     1 1.4E+00     5  4.277073E+01 1.2E-01    6 TF n
     3     4 2.3E+00     7  4.611470E+01 1.3E-01    5 TF n
     4     6 9.8E-01    10  4.759489E+01 8.0E-02    4 TF n
     5     3 1.9E-01    14  4.763815E+01 2.3E-02    4 TF n
     6     2 5.8E-01    16  4.767446E+01 2.9E-02    5 TF n
     7     1 5.0E-01    18  4.768810E+01 1.4E-02    5 TF n
     8     2 8.0E-01    20  4.770052E+01 9.0E-03    6 TF n
     9     1 8.4E-02    23  4.770088E+01 8.4E-03    6 TF n
    10     1 1.0E+00    24  4.770589E+01 2.5E-03    6 TF n

 Major Minor   Step   nObj   Objective   Optimal   nS PD
    11     1 9.3E-01    26  4.770648E+01 1.4E-03    6 TF n
    12     1 1.1E+00    28  4.770651E+01 6.2E-04    6 TF n
    13     1 1.4E+00    30  4.770651E+01 9.6E-05    6 TF n
    14     1 1.1E+00    32  4.770651E+01 4.1E-06    6 TF n
    15     1 9.5E-01    34  4.770651E+01 1.0E-07    6 TT n

 EXIT - Optimal Solution found, objective:       -47.70651

--- Restarting execution
--- chem.gms(49) 0 Mb
--- Reading solution for model mixer
*** Status: Normal completion
--- Job chem.gms Stop 10/31/06 16:34:47 elapsed 0:00:00.187
```

For a nonlinearly constrained problem, the log is somewhat different:

```
--- Job chenery.gms Start 10/31/06 16:38:28
GAMS Rev 146  Copyright (C) 1987-2006 GAMS Development. All rights reserved
*** ************* BETA release
*** GAMS Rev 146  BETA  1Nov06 WIN.00.NA 22.3 146.000.041.VIS P3PC
*** ************* BETA release
Licensee: Erwin Kalvelagen                              G060302/0001CE-WIN
          GAMS Development Corporation                              DC4572
--- Starting compilation
--- chenery.gms(242) 3 Mb
--- Starting execution
--- chenery.gms(224) 4 Mb
--- Generating NLP model chenrad
--- chenery.gms(227) 6 Mb
---   39 rows  44 columns  133 non-zeroes
---   390 nl-code  56 nl-non-zeroes
--- chenery.gms(227) 4 Mb
--- Executing SNOPT

SNOPT-Link    BETA  1Nov06 WIN.SN.SN 22.3 043.058.041.VIS SNOPT 7.2-4

    GAMS/SNOPT, Large Scale Nonlinear SQP Solver
    S N O P T  7.2-4 (June 2006)
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.29 Mb

  Reading Rows...
  Reading Columns...
  Reading Instructions...

   Itn     7 linear constraints are feasible, starting major iterations

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
     0    33 0.0E+00     1  6.000000E+02 1.4E+00 9.0E-03    5 0.0E+00 FF n r
     1     2 5.8E-03     2 -9.659720E+06 1.4E+00 1.0E+00    5 7.6E-01 FF n r
     2     1 6.3E-03     4 -5.124694E+05 1.4E+00 1.0E-01    5 7.6E-01 FF n r
     3    36 1.0E+00     6  4.088952E+04 9.8E-01 8.4E-02    5 7.6E-01 FF n r
     4     4 1.0E+00     8 -1.379500E+04 4.9E-01 1.4E-01    6 1.7E+00 FF n
     5     1 1.0E+00     9 -1.506550E+03 1.3E-01 7.3E-01    6 1.7E+00 FF n
     6     3 1.0E+00    11  4.635672E+02 2.2E-02 4.1E-01    4 1.7E+00 FF n
     7     6 1.5E-01    19  4.961587E+02 2.4E-02 5.9E-01    2 1.7E+00 FF n
     8     3 1.4E-01    26  5.306161E+02 2.4E-02 1.3E+00    1 1.7E+00 FF n
     9     2 1.4E-01    33  5.665326E+02 2.5E-02 1.4E+00    1 1.7E+00 FF n
    10     6 1.4E-01    40  6.039374E+02 2.5E-02 5.8E-01    1 1.7E+00 FF n

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    11     4 1.4E-01    47  6.398006E+02 2.4E-02 4.1E-01    1 1.7E+00 FF n
    12     9 1.8E-01    54  6.789740E+02 3.0E-02 7.8E-01    1 1.7E+00 FF n
    13     4 1.2E-01    61  7.058077E+02 3.0E-02 9.0E-01    1 1.7E+00 FF n
    14     6 1.1E-01    68  7.298681E+02 2.8E-02 1.3E-01    2 1.7E+00 FF n R
    15     2 6.7E-01    72  8.436769E+02 1.9E-02 1.1E-01    1 1.7E+00 FF n
    16     6 4.8E-01    74  8.963803E+02 9.8E-02 1.2E-01    2 1.7E+00 FF n
    17     5 4.8E-01    77  9.344485E+02 6.3E-02 2.0E-02    2 1.7E+00 FF n
    18     3 1.1E-01    84  9.433449E+02 5.8E-02 1.3E-02    0 1.7E+00 FF n
    19     1 1.2E-01    90  9.524431E+02 5.3E-02 1.2E-02    1 1.7E+00 FF n
    20     1 1.1E-01    97  9.599707E+02 4.8E-02 1.1E-02    1 1.7E+00 FF n
    21     1 1.0E-01   104  9.665613E+02 4.3E-02 1.1E-02    1 1.7E+00 FF n

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    22     1 9.8E-02   111  9.725279E+02 3.9E-02 1.4E-02    1 1.7E+00 FF n
    23     1 9.1E-02   119  9.779694E+02 3.5E-02 4.2E-02    1 1.7E+00 FF n
    24     1 8.6E-02   127  9.829387E+02 3.2E-02 7.3E-02    1 1.7E+00 FF n
    25     2 8.2E-02   135  9.874863E+02 3.0E-02 9.6E-03    2 1.7E+00 FF n R
    26     1 9.1E-01   137  1.015804E+03 1.9E-02 3.3E-02    2 1.7E+00 FF n
    27     2 1.0E+00   138  1.020587E+03 7.0E-03 9.4E-03    3 1.7E+00 FF n
    28     1 8.3E-01   140  1.020953E+03 1.8E-03 9.2E-03    3 1.7E+00 FF n
    29      2 1.0E+00   141   1.022140E+03 6.9E-05 1.2E-01    2 1.7E+00 FF n
    30      2 2.0E-01   146   1.026369E+03 2.1E-03 1.5E-01    3 1.7E+00 FF n
    31      1 1.6E-01   151   1.029081E+03 3.1E-03 1.5E-01    3 1.7E+00 FF n
    32      1 1.3E-01   157   1.031408E+03 3.6E-03 1.5E-01    3 1.7E+00 FF n

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    33      1 1.2E-01   163   1.033557E+03 4.0E-03 1.4E-01    3 1.7E+00 FF n
    34      1 1.1E-01   169   1.035594E+03 4.3E-03 1.3E-01    3 1.7E+00 FF n
    35      2 1.1E-01   175   1.037544E+03 4.6E-03 1.1E-01    2 1.7E+00 FF n
    36      2 1.3E-01   180   1.039584E+03 4.8E-03 3.6E-03    3 1.7E+00 FF n R
    37      1 1.0E+00   181   1.046222E+03 7.6E-05 3.5E-03    3 1.7E+00 FF n
    38      2 1.0E+00   182   1.046738E+03 7.1E-05 5.6E-02    2 1.7E+00 FF n
    39      2 5.1E-01   184   1.049069E+03 1.6E-03 4.6E-02    1 1.7E+00 FF n
    40      1 2.7E-01   188   1.050915E+03 2.4E-03 2.6E-02    1 1.7E+00 FF n
    41      2 4.3E-01   191   1.053326E+03 2.8E-03 1.5E-02    2 1.7E+00 FF n
    42      1 1.0E+00   192   1.057724E+03 2.1E-03 1.3E-03    2 1.7E+00 FF n
    43      1 1.0E+00   193   1.058918E+03 1.5E-05 2.9E-04    2 1.7E+00 FF n

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    44      1 1.0E+00   194   1.058920E+03 2.7E-06 1.6E-05    2 1.7E+00 FF n
    45      1 1.0E+00   195   1.058920E+03 3.7E-09 5.2E-07    2 1.7E+00 TT n

 EXIT - Optimal Solution found, objective:        1058.920

--- Restarting execution
--- chenery.gms(227) 0 Mb
--- Reading solution for model chenrad
--- Executing after solve
--- chenery.gms(241) 3 Mb
*** Status: Normal completion
--- Job chenery.gms Stop 10/31/06 16:38:29 elapsed 0:00:00.125
```

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 requested by user   --    0.76 Mb
   Work space requested by solver --    0.02 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 §2).

Step The step length α 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 (2.6). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see §2). 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 (4.8). 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 (4.9). 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 (4.8) and (4.9) 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(γ).
If SNOPT is already in elastic mode, an "`i`" indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

`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.

`m` 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.

`w` A weak solution of the QP subproblem was found.

Finally SNOPT prints an exit message. See §5.1.

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

$$\text{Max Dual infeas/Norm of pi} = 10^{-d},$$

then the objective function would probably change in the dth 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 $10e^{-2}$ 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 §4), 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)`
`EXIT -- Too many (minor) iterations`

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^THZ, 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:

```
               S O L V E     S U M M A R Y

     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-Link    BETA  1Nov06 WIN.SN.SN 22.3 043.058.041.VIS SNOPT 7.2-4

    GAMS/SNOPT, Large Scale Nonlinear SQP Solver
    S N O P T  7.2-4 (June 2006)
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University


 Work space allocated           --    0.29 Mb

 EXIT - Optimal Solution found, objective:        1058.920

 Major, Minor Iterations    45     179
 Funobj, Funcon calls      196     196
 Superbasics                 2
 Aggregations                0
 Interpreter Usage        0.00     0.0%

 Work space used by solver      --    0.11 Mb
```

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 34.1 and 34.2.

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 §4.1).

Model status	**Remarks**
1 OPTIMAL	Applies only to linear models.
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 INTERMEDIATE NONOPTIMAL	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 34.1: Model status values

Solver status	**Remarks**
1 NORMAL COMPLETION	SNOPT completed the optimization task.
2 ITERATION INTERRUPT	Iteration limit was hit. Increase the `iterlim` option (see §4.1).
3 RESOURCE INTERRUPT	Time limit was hit. Increase the `reslim` option (see §4.1).
4 TERMINATED BY SOLVER	Check the listing file.
5 EVALUATION ERROR LIMIT	`domlim` error limit was exceeded. See §4.1.
6 UNKNOWN ERROR	Check the listing file for error messages.
ERROR SETUP FAILURE	Id.
ERROR SOLVER FAILURE	Id.
ERROR INTERNAL SOLVER FAILURE	Id.
ERROR SYSTEM FAILURE	Id.

Table 34.2: Solver status values

The possible EXIT messages are listed in §5.1.

The statistics following the EXIT message are as follows.

Major, minor iterations. The number of major and minor iterations for this optimization task. Note that the number of minor iterations is the same as reported by ITERATION COUNT.

Funobj, Funcon calls. The number of times SNOPT evaluated the objective function $f(x)$ or the constraint functions $F_i(x)$ and their gradients. For a linearly constrained problem the number of funcon calls should be zero.

Superbasics. This is number of superbasic variables in the reported solution. See §2.4.

Aggregations. The number of equations removed from the model by the objective function recovery algorithm (see §2.1).

Interpreter usage. This line refers to how much time was spent evaluating functions and gradients. Due to the low resolution of the clock and the small size of this model, it was concluded that 100% of the time was spent inside the routines that do function and gradient evaluations. For larger models these numbers are more accurate.

SNOPT References

[1] A. R. Conn, *Constrained optimization using a nondifferentiable penalty function*, SIAM J. Numer. Anal., 10 (1973), pp. 760–779.

[2] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

[3] S. K. Eldersveld, *Large-scale sequential quadratic programming algorithms*, PhD thesis, Department of Operations Research, Stanford University, Stanford, CA, 1991.

[4] R. Fletcher, *An ℓ_1 penalty method for nonlinear constraints*, in Numerical Optimization 1984, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, eds., Philadelphia, 1985, SIAM, pp. 26–40.

[5] R. Fourer, *Solving staircase linear programs by the simplex method. 1: Inversion*, Math. Prog., 23 (1982), pp. 274–313.

[6] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM J. Optim., 12 (2002), pp. 979-1006.

[7] ——, *SNOPT: An SQP algorithm for large-scale constrained optimization*, SIAM Rev., 47 (2005), pp. 99-131.

[8] ——, *Users guide for SQOPT Version 7: Software for large-scale linear and quadratic programming*, Numerical Analysis Report 2006-1, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2006.

[9] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *User's guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming*, Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, CA, 1986.

[10] ——, *A practical anti-cycling procedure for linearly constrained optimization*, Math. Prog., 45 (1989), pp. 437–474.

[11] ——, *Some theoretical properties of an augmented Lagrangian merit function*, in Advances in Optimization and Parallel Computing, P. M. Pardalos, ed., North Holland, North Holland, 1992, pp. 101–128.

[12] ——, *Sparse matrix methods in optimization*, SIAM J. on Scientific and Statistical Computing, 5 (1984), pp. 562–589.

[13] ——, *Maintaining LU factors of a general sparse matrix*, Linear Algebra and its Applications, 88/89 (1987), pp. 239–270.

[14] B. A. Murtagh and M. A. Saunders, *Large-scale linearly constrained optimization*, Math. Prog., 14 (1978), pp. 41–72.

[15] ——, *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Math. Prog. Study, 16 (1982), pp. 84–117.

[16] ——, *MINOS 5.5 User's Guide*, Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, Revised 1998.

XA

Contents

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, subsituting 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

```
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:

```
<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 & 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:

```
SemiCont a , b ;
```

or to define semi-integer variables -

```
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 & 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 variable 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 by changed by setting the ”strategy” option in an XA options file.

Branch & Bound Strategy	**Description of Selection Criteria**
1	Proprietary method. Default value. Excellent strategy, also add priority to integer variable and try 1P for additional performance gains.
2	Minimum change in the objective function. This strategy has not been very successful at solving MIP problems.
3	Priority based upon column order. This strategy probably does not have must meaning because you typically do not set the column order in GAMS .
4	Column closest to its integer bound. This strategy tends to send a variable to its lower bounds.
6	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.
7	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.
8	Column farthest from its integer bound. Next to Strategies 1 and 6 this is probably the next choice to try. Using priorities and variation P is also suggested.
9	Column randomly selected, useful when solving very large problems. Priority values helpful in multi-period models.
10	Apparent smoothest sides on the polytope. Priorities helpful.

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`".

Variation	Effect
`A` (+100)	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.
`B`	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.
`C`	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.
`D`	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.
`P`	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.

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 that 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:

```
<modelname>.optfile = 1 ;
```

where `<modelname>` is the name of the model specified in the model statement. For instance:

```
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.

Option	Description	Default
`BASIS`	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 DualSimplex No. `file.ext`: The filename containing an 'advance basis' for restarting. Default file extension is SAV. `none`: No 'advance basis' is specified, but the 'final basis' is saved in the problem filename with an extension of SAV. `never`: No 'advance basis' is specified, and the final 'basis' is not saved.	`none`

Option	Description	Default
DUALSIMPLEX	By default, XA uses the dual simplex algorithm to restart from an advanced basis (e.g. on nodes in the B&B tree). If DualSimplex is set to No, it will use primal simplex instead.	Yes
FORCE	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.	No
LIMITSEARCH	See Section 5.3.	
MATLIST	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. `Var :` List columns in row. `Con :` List rows in columns. `Both:` Var and Con `None:` no listing	None
SET BARRIER	Activates XA's primal-dual interior point algorithm. Useful when solving very large-scale LP models. `Yes:` Uses primal-dual interior point algorithm, MIP models automatically crossover the simplex algorithm for branching & bounding. `No :` Use the simplex algorithm. `X :` Crossover to the simplex algorithm after solving. (automatic when solving MIP models.	
SET BELL	Turns XA's termination bell on or off. `No :` do not ring the bell `Yes:` ring the bell	No
SET BVPRIORITY	Sets the default priority of all binary variables to #. By default, all variables have priority 1600, so a value < 1600 causes binary variables to be branched on before general integer variables. A value > 1600 has the opposite affect.	1600
SET CRASH	Method of generating initial basis. `0:` Minimize primal infeasibility `1:` Minimize dual infeasibility. `2:` Both 0 & 1.	0
SET DEGENITER	Degenerate anticycling aide. Number of consecutive degenerate pivots before anti-cycling code is activated.	square root of the number of rows.
SET FREQLOG	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.	00:01 (one log line per second).
SET INTGAP	Minimum objective function improvement between each new integer solutions. Reported integer solution may not be the optimal integer solution because of premature termination.	0.00
SET INTLIMIT	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.	no limit on the number of integer solutions.
SET INTPCT	Percentage of available integer columns to consider fixing at each integer node. 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.	0.0 - meaning no fixing.

Option	Description	Default
`SET IROUND`	XA reports either rounded or unrounded integer column primal activity. `Yes:` causes XA to report rounded integer column activity. XA rounds integer activity values to the closest bound based upon the LTOLERANCE and UTOLERANCE values. `No :` causes XA to report unrounded integer variable activity, but these activities will always be within the requested integer tolerance.	`Yes`
`SET ITERATION`	Maximum number of iteration. 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.	`2000000000`

`SET LTOLERANCE`
`SET UTOLERANCE`
	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 * Y is in the range [3 * 3.95, 3 * 4.001].	`5.0e-6`

`SET MARKOWITZ`	Numeric Stability vs. sparsity in basis updating and inverse. A larger number favors sparsity at the expense of numeric stability.	`10`
`SET MAXCPU`	Number of processors to use when solving MIP models. 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.	
`SET MAXNODES`	Memory estimate for number of branch and bound nodes. Default value: 4,000 plus the number of binary variables plus square root of the number of integer columns.	
`SET NODELIMIT`	Maximum number of branch and bound nodes. Default value: unlimited.	
`SET PERTURBATE`	Indicates the amount of perturbation for 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.	`0`, indicating no perturbation value
`SET PRICING`	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.	`0`
`SET REDUCEDCOST`	Dual activity tolerance to zero.	`1e-7`
`SET REINVERTFREQ`	The basis update factors are thrown away and the basis reinverted with this frequency.	`40`
`SET RELAXED`	Integer problem is solved as a standard LP and with no integer columns in the formulation. `No :` integer problems are solved with branch and bound method `Yes:` solve problems as if all columns where continuous columns	`No`

Option	Description	Default
SET RESTART	When solving integer, binary, or semi-continuous problems XA may terminate before exploring all your integer nodes. If so you have the option of restarting XA and picking up right where you left off. Just before XA terminates, it writes out basis information in the basis file (extension sav). If an integer, binary or semi-continuous solution has been found an integer basis file is created (extension b01). And if all your integer nodes are not explored then a third file is created for restarting the unchanged problem (extension r01). The BASIS command line parameter determines the filename used with these extensions. `Yes:` if an r01 file exists with my identical problem then restart integer problem where I left off and if XA terminates before all nodes are explored again then write restart information to this file `No :` do not use or create the r01 file to restart my problem	`No`
SET SCALE	Problem Scaling technique. `No :` Data not scaled. `Yes:` Column and row scaling. `2 :` Row scaling only.	`Yes`
SET STICKWITHIT	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 & bound strategy is used.	`10`
SET TCTOLERANCE	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 1.0e-15.	`1.0e-7`
SET TIMELIMIT	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.	`2000000000` seconds
SET YPIVOT	When selecting a column to leave the basis, columns with absolute marginal values less than ypivot are rejected. Pivoting in columns with very small values can lead to numeric instability and should be avoided when possible. Setting 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.	`1e-9`
SET XTOZERO	Primal activity tolerance to zer.o	`1e-7`
STOPAFTER	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.	`0` (indicates no termination)
STRATEGY	MIP search strategy.	`1`
TOMPS	Write an MPS file of problem. *gams.mps* file is created or rewritten. `No:` Do not write an MPS formatted file (default value). `Yes:` Write problem in MPS format. `Secure:` Write problem in MPS format and change column and row names to: C0, C1,... and R0, R1, ...	`No`

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
  ####    ###      ########     ########     #######     #######     #  #######  # ####
```

	Description
`Node`	Active node, the smaller the better, value increases and decreases as the branch-and-bound proceeds.
`IInf`	Number of discrete columns having fractional values. This number converges to 0 as XA approaches an integer solution.
`ToGo.Map`	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.
`Best.Obj`	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.
`Cur.Obj`	Objective function for the current node. If an integer solution is found in this node cannot be any better than this value.
`Int.Obj`	Objective function of the best integer solution found so far. This value improves as additional integer solutions are found.
`#`	Number of improving integer solutions found thus far.
`Column`	Column selected by the branch-and-bound process.
`+/-`	Branching direction: up(+) or down(-).
`Iter`	Cumulative total of simplex iterations used (including the relaxed LP).

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.

XPRESS

Contents

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.

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 request a particular model to be solved by XPRESS by inserting the statement

`option LP = xpress;` { or `MIP` or `RMIP` }

somewhere before the `solve` statement.

The following standard GAMS options can be used to control XPRESS-MP:

- `option reslim =x;` or `modelname.reslim = x;`

  Stop the algorithm after `x` seconds and report the best solution found so far. `Modelname` is the name of the model as specified in a previous `model` statement.

[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.

- `option iterlim=n;` or `modelname.iterlim = n;`

  Stop the algorithm after n simplex iterations and report the best solution found so far. For MIP models, this places a cumulative limit on simplex iterations for the relaxed LP and the nodes of the B&B tree. `Modelname` is the name of the model as specified in a previous `model` statement.

- `option sysout=on;`

  Echo more detailed information about the solution process to the listing file.

- `option optcr=x;`

  In a MIP stop the search as soon as the relative gap is less than `x`.

- `option optca=x;`

  In a MIP stop the search as soon as the absolute gap is less than `x`.

- `option bratio=x;`

  Determines whether or not an advanced basis is passed on to the solver. `Bratio=1` will always ignore an existing basis (in this case XPRESS-MP will use a crash routine to find a better basis than an all-slack basis), while `bratio=0` will always accept an existing basis. Values between 0 and 1 use the number of non-basic variables found to determine if a basis is likely to be good enough to start from.

- `modelname.prioropt = 1;`

  Turns on usage of user-specified priorities. Priorities can be assigned to integer and binary variables using the syntax: `variablename.prior = x;`. Default priorities are 0.0. Variables with a priority v1 are branched upon earlier than variables with a priority `v2` if `v1<v2`.

- `modelname.nodlim = n;`

  Specifies a node limit for the Branch-and-Bound search. When the number of nodes exceeds this number the search is stopped and the best integer solution found (if any) is reported back to GAMS. The default value of 0 indicates: no node limit.

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.

3 Options

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:

```
* 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 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

The following general options control the behavior of the GAMS/XPRESS link.

Option	Description	Default
`advbasis`	`0:` don't use advanced basis provided by GAMS `1:` use advanced basis provided by GAMS This option overrides the GAMS BRATIO option.	Determined by GAMS
`algorithm`	`simplex:` use simplex solver `barrier:` use barrier algorithm This option is used to select the barrier method to solve LP's. By default the barrier method will do a crossover to find a basic solution.	`simplex`
`basisout`	If specified an MPS basis file is written. In general this option is not used in a GAMS environment, as GAMS maintains basis information for you automatically.	Don't write a basis file.
`iterlim`	Sets the iteration limit for simplex algorithms. When this limit is reached the system will stop and report the best solution found so far. Overrides the GAMS ITERLIM option.	`10000`
`mpsoutputfile`	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.	Don't write an MPS file.
`reform`	If true, the link will try to reformulate the model by removing the objective variable and the equation it appears in and replacing these with an objective function. If false, or if reformulation is not possible, the objective variable is not removed and the problem passed to the optimizer will have a very simple objective: the objective variable.	`1`
`rerun`	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.	`1`
`reslim`	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. Overrides the GAMS RESLIM option.	`1000.0`

The following general options set XPRESS library control variables, and can be used to fine-tune XPRESS.

Option	Description	Default
`crash`	A crash procedure is used to quickly find a good basis. This option is only relevant when no advanced basis is available. `0:` no crash `1:` singletons only (one pass) `2:` singletons only (multi-pass) `3:` multiple passes through the matrix considering slacks `4:` multiple passes (<= 10), but do slacks at the end `>10:` as 4 but perform n-10 passes `100:` default crash behavior of XPRESS-MP version 6	`0` when GAMS provides an advanced basis, and `2` otherwise

Option	Description	Default
`extrapresolve`	The initial number of extra elements to allow for in the presolve. 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.	`automatic`
`lpiterlimit`	Sets the iteration limit for simplex algorithms. For MIP models, this is a per-node iteration limit for the B&B tree. Overrides the iterlim option.	`MAXINT`
`mpsnamelength`	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.	`0`
`presolve`	`-1`: Presolve applied, but a problem will not be declared infeasible if primal infeasibilities are detected. The problem will be solved by the LP optimization algorithm, returning an infeasible solution, which can sometimes be helpful. `0`: Presolve not applied. `1`: Presolve applied. `2`: Presolve applied, but redundant bounds are not removed. This can sometimes increase the efficiency of the barrier algorithm. As XPRESS-MP does a basis preserving presolve, you don't have to turn off the presolver when using an advanced basis.	`1`
`scaling`	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. `Bit 0 = 1`: Row scaling. `Bit 1 = 2`: Column scaling. `Bit 2 = 4`: Row scaling again. `Bit 3 = 8`: Maximin. `Bit 4 = 16`: Curtis-Reid. `Bit 5 = 32`: Off implies scale by geometric mean, on implies scale by maximum element. Not applicable for maximin and Curtis-Reid scaling.	`35`
`threads`	Controls the number of threads to use. Positive values will be compared to the number of available cores detected and reduced if greater than this amount. 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.	`1`
`trace`	Control of the infeasibility diagnosis during presolve - if nonzero, infeasibility will be explained.	`0`

3.2 LP Options

The following options set XPRESS library control variables, and can be used to fine-tune the XPRESS LP solver.

Option	Description	Default
`bigmmethod`	`0:` for phase I / phase II `1:` if 'big M' method to be used	automatic
`bigm`	The infeasibility penalty used in the "Big M" method	automatic
`defaultalg`	`1:` automatic `2:` dual simplex `3:` primal simplex `4:` Newton barrier	`1`
`etatol`	Zero tolerance on eta elements. During each iteration, the basis inverse is premultiplied by an elementary matrix, which is the identity except for one column the eta vector. Elements of eta vectors whose absolute value is smaller than `etatol` are taken to be zero in this step.	`1.0e-12`
`feastol`	This is the zero tolerance on right hand side values, bounds and range values. If one of these is less than or equal to `feastol` in absolute value, it is treated as zero.	`1.0e-6`
`invertfreq`	The frequency with which the basis will be inverted. A value of -1 implies automatic.	`-1`
`invertmin`	The minimum number of iterations between full inversions of the basis matrix.	`3`
`lplog`	The frequency at which the simplex iteration log is printed. `n < 0:` detailed output every -n iterations `n = 0:` log displayed at the end of the solution process `n > 0:` summary output every n iterations	`100`
`matrixtol`	The zero tolerance on matrix elements. If the value of a matrix element is less than or equal to `matrixtol` in absolute value, it is treated as zero.	`1.0e-9`
`optimalitytol`	This is the zero tolerance for reduced costs. On each iteration, the simplex method searches for a variable to enter the basis which has a negative reduced cost. The candidates are only those variables which have reduced costs less than the negative value of `optimalitytol`.	`1.0e-6`
`penalty`	Minimum absolute penalty variable coefficient used in the "Big M" method.	automatic
`pivottol`	The zero tolerance for matrix elements. On each iteration, the simplex method seeks a nonzero matrix element to pivot on. Any element with absolute value less than `pivottol` is treated as zero for this purpose.	`1.0e-9`
`pricingalg`	This determines the pricing method to use on each iteration, selecting which variable enters the basis. In general Devex pricing requires more time on each iteration, but may reduce the total number of iterations, whereas partial pricing saves time on each iteration, although possibly results in more iterations. `-1:`if partial pricing is to be used `0:` if the pricing is to be decided automatically. `1:` if DEVEX pricing is to be used	`0`
`relpivottol`	At each iteration a pivot element is chosen within a given column of the matrix. The relative pivot tolerance, `relpivottol`, is the size of the element chosen relative to the largest possible pivot element in the same column.	`1.0e-6`

3.3 MIP Options

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:

I `iterlim` (on the cumulative pivot count), `reslim` (in seconds of CPU time),

II `optca` & `optcr` (stopping criteria based on gap between best integer solution found and best possible) or

III `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 the table of XPRESS control variables for MIP below.

The following options control the behavior of the GAMS/XPRESS link on MIP models.

Option	**Description**	**Default**
`loadmipsol`	If true, the initial point provided by GAMS will be passed to the optimizer *to be treated as an integer feasible point.* The optimizer uses the values for the discrete variables only: the values for the continuous variables are ignored and are calculated by fixing the integer variables and reoptimizing. In some cases, loading an initial MIP solution can improve performance. In addition, there will always be a feasible solution to return.	`0`
`mipcleanup`	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.	`1`
`miptrace`	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.	none
`miptracenode`	Specifies the node interval between entries to the `miptrace` file.	`100`
`miptracetime`	Specifies the time interval, in seconds, between entries to the `miptrace` file.	`5`

The following options set XPRESS library control variables, and can be used to fine-tune the XPRESS MIP solver.

Option	**Description**	**Default**
`backtrack`	This determines how the next node in the tree search is selected for processing. `1`: If `miptarget` is not set, choose the node with the best estimate. If `miptarget` is set (by the user or by the global search previously finding an integer solution), the choice is based on the Forrest-Hirst-Tomlin Criterion, which takes into account the best current integer solution and seeks a new node which represents a large potential improvement. `2`: Always choose the node with the best estimated solution. `3`: Always choose the node with the best bound on the solution.	`3`
`breadthfirst`	If `nodeselection` = 4, this determines the number of nodes to include in a breadth-first search.	`10`
`covercuts`	The 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.	`20`
`cpmaxcuts`	The initial maximum number of cuts that will be stored in the cut pool. During optimization, the cut pool is subsequently resized automatically.	`100`
`cpmaxelems`	The initial maximum number of nonzero coefficients which will be held in the cut pool. During optimization, the cut pool is subsequently resized automatically.	`200`
`cutdepth`	Sets the maximum depth in the tree search 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.	`0`
`cutfreq`	This specifies the frequency at which cuts are generated in the tree search. If the depth of the node modulo `cutfreq` is zero, then cuts will be generated.	`8`

Option	Description	Default
`cutstrategy`	This 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. `-1:` Automatic selection of either the conservative or aggressive cut strategy. `0:` No cuts. `1:` Conservative cut strategy. `2:` Aggressive cut strategy.	`-1`
`degradefactor`	Factor to multiply estimated degradations associated with an unexplored node in the tree. The estimated degradation is the amount by which the objective function is expected to worsen in an integer solution that may be obtained through exploring a given node.	`1.0`
`gomcuts`	The 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.	`2`
`maxmipsol`	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.	`0`
`maxnode`	The maximum number of nodes that will be explored. If the GAMS nodlim model suffix is set, that setting takes precedence.	`1e8`
`mipabscutoff`	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 `mipabscutoff`. This allows the Optimizer to ignore solving any nodes which may yield worse objective values, saving solution time.	automatic
`mipaddcutoff`	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.	`+/-1.0e-5`
`miplog`	Global print control `0:` No printout in global. `1:` Only print out summary statement at the end. `2:` Print out detailed log at all solutions found. `3:` Print out detailed eight-column log at each node. `n < 0:` Print out summary six-column log at each -n nodes, or when a new solution is found	`-100`

Option	Description	Default
`mippresolve`	Bitmap determining type of integer processing to be performed. If set to 0, no processing will be performed. `Bit 0:` Reduced cost fixing will be performed at each node. This can simplify the node before it is solved, by deducing that certain variables' values can be fixed based on additional bounds imposed on other variables at this node. `Bit 1:` Logical preprocessing will be performed at each node. This is performed on binary variables, often resulting in fixing their values based on the constraints. This greatly simplifies the problem and may even determine optimality or infeasibility of the node before the simplex method commences. `Bit 2:` Probing of binary variables is performed at the top node. This sets certain binary variables and then deduces effects on other binary variables occurring in the same constraints.	automatic
`miprelcutoff`	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.	`1.0e-4`
`miptarget`	The target objective value for integer solutions. This is automatically set after solving the LP unless set by the user.	`1e40`
`miptol`	This is the tolerance within which a decision variable's value is considered to be integral.	`5.0e-6`
`nodeselection`	This determines which nodes will be considered for solution once the current node has been solved. `1:` *Local first*: Choose among the two descendant nodes, if none among all active nodes. `2:` *Best first*: All nodes are always considered. `3:` *Depth first*: Choose deepest node, but explore two descendents first. `4:` *Best first, then local first*: All nodes are considered for the first `breadthfirst` nodes, after which the usual default behavior is resumed.	automatic
`pseudocost`	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.	`0.01`
`sleepOnThreadWait`	Threads during a MIP solve can now busy-wait instead of going to sleep when waiting for work. This is to overcome a performance issue with modern speed-stepping CPUs, which might step down to a lower clock frequency when the load is less than 100%.	`no`
`treecovercuts`	The number of rounds of lifted cover inequalities generated at nodes other than the top node in the tree. Compare with the description for `covercuts`.	`2`
`treegomcuts`	The number of rounds of Gomory cuts generated at nodes other than the first node in the tree. Compare with the description for `gomcuts`.	`0`
`varselection`	This determines how to combine the pseudo costs associated with the integer variables to obtain an overall estimated degradation in the objective function that may be expected in any integer solution from exploring a given node. It is relevant only if `backtrack` has been set to 1. `1:` Sum the minimum of the 'up' and 'down' pseudo costs. `2:` Sum all of the 'up' and 'down' pseudo costs. `3:` Sum the maximum, plus twice the minimum of the 'up' and 'down' pseudo costs. `4:` Sum the maximum of the 'up' and 'down' pseudo costs. `5:` Sum the 'down' pseudo costs. `6:` Sum the 'up' pseudo costs.	`1`

3.4 Newton-Barrier Options

The barrier method is invoked 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.

The following options set XPRESS library control variables, and can be used to fine-tune the XPRESS barrier solver.

Option	Description	Default
`bariterlimit`	Maximum number of barrier iterations	`200`
`crossover`	Determines whether the barrier method will cross over to the simplex method when at optimal solution has been found, in order to provide an end basis. `0`: No crossover. `1`: Crossover to a basic solution.	`1`

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.