GUROBI 5.1

Gurobi Optimization, `www.gurobi.com`

Contents

1 Introduction

The Gurobi suite of optimization products include state-of-the-art simplex and parallel barrier solvers for linear programming (LP) and quadratic programming (QP), parallel barrier solver for quadratically constrained programming (QCP), as well as parallel mixed-integer linear programming (MILP), mixed-integer quadratic programming (MIQP) and mixed-integer quadratically constrained programming (MIQCP) solvers.

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, Quadratic and Quadratic Constrained Programming

Gurobi can solve LP and QP problems using several alternative algorithms, while the only choice for solving QCP is the parallel barrier algorithm. The majority of LP problems solve best using Gurobi's state-of-the-art dual simplex algorithm, while most QP problems solve best using the parallel barrier algorithm. Certain types of LP problems benefit from using the parallel barrier or the primal simplex algorithms, while for some types of QP, the dual or primal simplex algorithm can be a better choice. If you are solving LP problems on a multi-core system, you should also consider using the concurrent optimizer. It runs different optimization algorithms on different cores, and returns when the first one finishes.

GAMS/Gurobi also provides access to the Gurobi infeasibility finder. The infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Gurobi reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing. The infeasibility finder is activated by the option iis. Another option for analyzing infeasible model the FeasOpt option which intructs GAMS/Gurobi to find a minimal feasible relaxation of an infeasible model. See section 3.3 for details.

GAMS/Gurobi supports sensitivity analysis (post-optimality analysis) for linear programs which allows one to find out more about an optimal solution for a problem. In particular, objective ranging and constraint ranging give information about how much an objective coefficient or a right-hand-side and variable bounds can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or the bounds and right-hand side. GAMS/Gurobi reports the sensitivity information as part of the normal solution listing. Sensitivity analysis is activated by the option sensitivity.

The Gurobi presolve can sometimes diagnose a problem as being infeasible *or* unbounded. When this happens, GAMS/Gurobi can, in order to get better diagnostic information, rerun the problem with presolve turned off. The rerun without presolve is controlled by the option rerun. In default mode only problems that are small (i.e. demo sized) will be rerun.

Gurobi can either presolve a model or start from an advanced basis or primal/dual solution pair. Often the solve from scratch of a presolved model outperforms a solve from an unpresolved model started from an advanced basis/solution. It is impossible to determine a priori if presolve or starting from a given advanced basis/solution without presolve will be faster. By default, GAMS/Gurobi will automatically use an advanced basis or solution from a previous `solve` statement. The GAMS *BRatio* option can be used to specify when not to use an advanced basis/solution. The GAMS/Gurobi option usebasis can be used to ignore or force a basis/solution passed on by GAMS (it overrides *BRatio*). In case of multiple solves in a row and slow performance of the second and subsequent solves, the user is advised to set the GAMS *BRatio* option to 1.

3.2 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear or quadratic programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with discrete variables, Gurobi uses a branch and cut algorithm which solves a series of subproblems, LP subproblems for MILP, QP subproblems for MIQP, and QCP subproblems or LP outer approximation subproblems for MIQCP. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS/Gurobi supports Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution.

If you specify some or all values for the discrete variables together with GAMS/Gurobi option mipstart, Gurobi will check the validity of the values as an integer-feasible solution. If this process succeeds, the solution will be treated as an integer solution of the current problem.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

3.3 Feasible Relaxation

The Infeasibility Finder identifies the causes of infeasibility by means of inconsistent set of constraints (IIS). However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Gurobi is known as FeasOpt (for Feasible Optimization) and turned on by parameter FeasOpt in a GAMS/Gurobi option file.

With the FeasOpt option GAMS/Gurobi accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are 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. The input value denotes the users willingness to relax a constraint or bound. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed. It is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a GAMS/Gurobi solver option file. The syntax is:

(variable or equation)`.feaspref` *(value)*

For example, suppose we have a GAMS declaration:

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

Then, the relaxation preference in the *gurobi.opt* file can be specified by:

```
feasopt 1
v.feaspref              1
v.feaspref('i1',*)      2
v.feaspref('i1','j2')   0

e.feaspref(*,'j1')      0
e.feaspref('i5','j4')   2
```

First we turn the feasible 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 FeasOptMode for details. Also check example models `feasopt*` in the GAMS Model library.

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`.

***ModelName*.PriorOpt = 1;**

Instructs GAMS/Gurobi to use the priority branching information passed by GAMS through variable suffix values *variable*`.prior`.

5 Summary of GUROBI Options

5.1 Termination options

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
barqcpconvtol	Convergence tolerance for the barrier algorithm when solving a QCP
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 options

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
sifting	Sifting within dual simplex
siftmethod	LP method used to solve sifting sub-problems
simplexpricing	Determines variable pricing strategy

5.4 Barrier options

barcorrectors	Limits the number of central corrections performed in each barrier iteration
barhomogeneous	
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
qcpdual	Determines whether dual variable values are computed for QCP models

5.5 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
improvestartnodes	Solution improvement strategy control
improvestarttime	Elapsed time after which the MIP solver resets a few MIP parameters
miqcpmethod	Determines whether outer approximation is used to solve an MIQCP model.
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
presparsify	Enables the presolve sparsify reduction for MIP models
.prior	Branching priorities
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
zeroobjnodes	Number of nodes to explore in the zero objective heuristic

5.6 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
feasopt	Computes a minimum-cost relaxation to make an infeasible model feasible
feasoptmode	Mode of FeasOpt
.feaspref	feasibility preference
fixoptfile	Option file for fixed problem optimization
iis	Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible
iismethod	Controls use of IIS method
kappa	Display approximate condition number estimates for the optimal simplex basis
kappaexact	Display exact condition number estimates for the optimal simplex basis

method	Algorithm used to solve continuous models
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
preqlinearize	Controls linearization of Q matrices in the quadratic constraints or a quadratic objective
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
seed	Random number seed
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

5.7 The GAMS/Gurobi Options File

The GAMS/Gurobi options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file *gurobi.opt*.

```
simplexpricing 3
method 0
```

It will cause Gurobi to use quick-start steepest edge pricing and will use the primal simplex algorithm.

6 GAMS/Gurobi Log File

Gurobi reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the simplex algorithms, each log line starts with the iteration number, followed by the objective value, the primal and dual infeasibility values, and the elapsed wall clock time. The dual simplex uses a bigM approach for handling infeasibility, so the objective and primal infeasibility values can both be very large during phase I. The frequency at which log lines are printed is controlled by the *displayinterval* option. By default, the simplex algorithms print a log line roughly every five seconds, although log lines can be delayed when solving models with particularly expensive iterations.

The simplex screen log has the following appearance:

```
Presolve removed 977 rows and 1539 columns
Presolve changed 3 inequalities to equalities
Presolve time: 0.078000 sec.
Presolved: 1748 Rows, 5030 Columns, 32973 Nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
```

```
       0    3.8929476e+31   1.200000e+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)

barhomogeneous (*integer)* Determines whether to use the homogeneous barrier algorithm. At the default setting (-1), it is only used when barrier solves a node relaxation for a MIP model. Setting the parameter to 0 turns it off, and setting it to 1 forces it on. The homogeneous algorithm is useful for recognizing infeasibility or unboundedness. It is a bit slower than the default algorithm.

(default = -1)

-1	Auto
0	Homogeneous Barrier off
1	Force Homogeneous Barrier on

barorder (*integer)* Chooses the barrier sparse matrix fill-reducing algorithm

(default = -1)

-1	Auto
0	Approximate Minimum Degree ordering
1	Nested Dissection ordering

barqcpconvtol (*real)* Convergence tolerance for the barrier algorithm when solving a QCP

When solving a QCP model, the barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. Tightening this tolerance may lead to a more accurate solution, but it may also lead to a failure to converge.

(default = 1e-6)

branchdir (*integer)* Determines which child node is explored first in the branch-and-cut search

This option allows more control over how the branch-and-cut tree is explored. Specifically, when a node in the MIP search is completed and two child nodes, corresponding to the down branch and the up branch are created, this parameter allows you to determine whether the MIP solver will explore the down branch first, the up branch first, or whether it will choose the next node based on a heuristic determination of which sub-tree appears more promising.

(default = 0)

-1	Always explore the down branch first
0	Automatic

1 Always explore the up branch first

cliquecuts (*integer)* Controls clique cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

-1 Auto

0 Off

1 Conservative

2 Aggressive

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)

feasopt (*integer*) Computes a minimum-cost relaxation to make an infeasible model feasible

With `Feasopt` turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option feasoptmode.

Feasible relaxations are available for all problem types.

(default = 0)

feasoptmode (*integer*) Mode of FeasOpt

The parameter `FeasOptMode` allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter *FeasOptMode* indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations).

(default = 0)

0 Minimize sum of relaxations. Minimize the sum of all required relaxations in first phase only

1 Minimize sum of relaxations and optimize. Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations

2 Minimize number of relaxations. Minimize the number of constraints and bounds requiring relaxation in first phase only

3 Minimize number of relaxations and optimize. Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations

4 Minimize sum of squares of relaxations. Minimize the sum of squares of required relaxations in first phase only

5 Minimize sum of squares of relaxations and optimize. Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref (*real*) feasibility preference

You can express the costs associated with relaxing a bound or right hand side value during a `feasopt` run through the `.feaspref` option. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

(default = 1)

fixoptfile (*string*) Option file for fixed problem optimization

flowcovercuts (*integer*) Controls flow cover cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

-1 Auto

0 Off

1 Conservative

2 Aggressive

flowpathcuts (*integer*) Controls flow path cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

-1 Auto

0 Off

1 Conservative

2 Aggressive

gomorypasses (*integer*) Maximum number of Gomory cut passes

A non-negative value indicates the maximum number of Gomory cut passes performed. See the description of the global Cuts parameter for further information.

(default = -1)

gubcovercuts (*integer*) Controls GUB cover cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

-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 Irreducible Inconsistent Subsystem (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)

improvestartnodes (*real)* Solution improvement strategy control

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify the node count at which the MIP solver switches to a solution improvement strategy. For example, setting this parameter to 10 will cause the MIP solver to switch strategies once the node count is larger than 10.

(default = maxdouble)

improvestarttime (*real)* Elapsed time after which the MIP solver resets a few MIP parameters

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify a time limit when the MIP solver will switch to this strategy. For example, setting this parameter to 10 will cause the MIP solver to switch 10 seconds after starting the optimization.

(default = maxdouble)

intfeastol (*real)* Integer feasibility tolerance

An integrality restriction on a variable is considered satisfied when the variable's value is less than *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 approximate condition number estimates for the optimal simplex basis

(default = 0)

kappaexact (*integer)* Display exact condition number estimates for the optimal simplex basis

(default = 0)

markowitztol (*real*) Threshold pivoting tolerance

Used to limit numerical error in the simplex algorithm. A larger value may avoid numerical problems in rare situations, but it will also harm performance.

Range: [1e-4,0.999]

(default = 0.0078125)

method (*integer*) Algorithm used to solve continuous models

Concurrent optimizers run multiple solvers on multiple threads simultaneously, and choose the one that finishes first. Deterministic concurrent (4) gives the exact same result each time, while concurrent (3) is often faster but can produce different optimal bases when run multiple times. In the current release, the default Automatic (-1) will choose non-deterministic concurrent (3) for an LP, barrier (2) for a QP, and dual (1) for the MIP root node. Only simplex and barrier algorithms are available for continuous QP models. Only primal and dual simplex are available for solving the root of an MIQP model. Only barrier is available for continuous QCP models.

The default setting is rarely significantly slower than the best possible setting, so you generally won't see a big gain from changing this parameter. There are classes of models where one particular algorithm is consistently fastest, though, so you may want to experiment with different options when confronted with a particularly difficult model.

Note that if memory is tight on an LP model, you should consider choosing the dual simplex method (`Method=1`). The default will invoke the concurrent optimizer, which typically consumes a lot more memory than dual simplex alone.

(default = -1)

-1	Automatic
0	Primal simplex
1	Dual simplex
2	Barrier
3	Concurrent
4	Deterministic concurrent

miqcpmethod (*integer*) Determines whether outer approximation is used to solve an MIQCP model.

Controls the method used to solve MIQCP models. Value 1 uses a linearized, outer-approximation approach, while value 0 solves continuous QCP relaxations at each node. The default setting (-1) chooses automatically.

(default = -1)

-1	Auto
0	Linearized, outer-approximation approach
1	Continuous QCP relaxations at each node

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)

preqlinearize (*integer*) Controls linearization of Q matrices in the quadratic constraints or a quadratic objective

Option 1 attempts to linearize quadratic constraints or a quadratic objective, potentially transforming an MIQP or MIQCP into an MILP. Option 0 shuts off the transformation. The default setting (-1) choose automatically. The automatic setting works well, but there are cases where forcing Q linearization can be beneficial.

(default = -1)

-1 Auto

0 Linearization off

1 Force Linearization on

presolve (*integer*) Controls the presolve level

(default = -1)

-1 Auto

0 Off

1 Conservative

2 Aggressive

presparsify (*integer*) Enables the presolve sparsify reduction for MIP models

This reduction can sometimes significantly reduce the number of nonzero values in the presolved model.

(default = 0)

printoptions (*integer*) List values of all options to GAMS listing file

(default = 0)

.prior (*real*) Branching priorities

GAMS allows to specfiy priorities for discrete variables only. Gurobi can detect that continuous variables are implied discrete variables and can utilize priorities. Such priorities can be specified through a GAMS/Gurobi solver option file. The syntax for *dot* options is explained in the Introduction chapter of the Solver Manual. The priorities are only passed on to Gurobi if the model attribute `priorOpt` is turned on.

(default = 1)

psdtol (*real)* limit on the amount of diagonal perturbation

Positive semi-definite tolerance (for QP/MIQP). Sets a limit on the amount of diagonal perturbation that the optimizer is allowed to automatically perform on the Q matrix in order to correct minor PSD violations. If a larger perturbation is required, the optimizer will terminate stating the problem is not PSD.

Range: [0,maxdouble]

(default = 1e-6)

pumppasses (*integer)* Number of passes of the feasibility pump heuristic

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic found a feasible solution.

(default = 0)

qcpdual (*integer)* Determines whether dual variable values are computed for QCP models

Determines whether dual variable values are computed for QCP models. Computing them can add significant time to the optimization, so you should turn this parameter to 0 if you do not need them.

(default = 1)

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)

seed (*integer)* Random number seed

Modifies the random number seed. This acts as a small perturbation to the solver, and typically leads to different solution paths.

(default = 0)

sensitivity (*integer)* Provide sensitivity information

(default = 0)

sifting (*integer)* Sifting within dual simplex

Enables or disables sifting within dual simplex. Sifting is often useful for LP models where the number of variables is many tiems larger than the number of constraints. With a *Moderate* setting, sifting will be applied to LP models and to the root node for MIP models. With an *Aggressive* setting, sifting will be also applied to the nodes of a MIP. Note that this parameter has no effect if you aren't using dual simplex. Note also that sifting will be skipped in cases where it is obviously a worse choice, even when sifting has been selected.

(default = -1)

-1	Auto
0	Off
1	Moderate
2	Agressive

siftmethod (*integer*) LP method used to solve sifting sub-problems

Note that this parameter only has an effect when you are using dual simplex and sifting has been selected (either by the automatic method, or through the *Sifting* parameter).

(default = -1)

-1	Auto
0	Primal Simplex
1	Dual Simplex
2	Barrier

simplexpricing (*integer*) Determines variable pricing strategy

(default = -1)

-1	Auto
0	Partial Pricing
1	Steepest Edge
2	Devex
3	Quick-Start Steepest Edge

solutionlimit (*integer*) Limits the number of feasible solutions found

(default = maxint)

solvefixed (*integer*) Indicator for solving the fixed problem for a MIP to get a dual solution

(default = 1)

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

If usebasis is not specified, GAMS (option bratio) decides if the starting basis or a primal/dual solution is given to Gurobi. If usebasis is excplictely set in an option file then the basis or a primal/dual solution is passed to Gurobi independent of the GAMS option bratio. Please note, if Gurobi uses a starting basis presolve will be skipped.

(default = GAMS bratio)

0	No basis
1	Supply basis if basis is full otherwise provide primal dual solution
2	Supply basis iff basis is full
3	Supply primal dual solution

varbranch (*integer)* Controls the branch variable selection strategy

(default = -1)

-1	Auto
0	Pseudo Reduced Cost Branching
1	Pseudo Shadow Price Branching
2	Maximum Infeasibility Branching
3	Strong Branching

writeparams (*string)* Write Gurobi parameter file

writeprob (*string)* Save the problem instance

zerohalfcuts (*integer)* Controls zero-half cut generation

See the description of the global Cuts parameter for further information.

(default = -1)

-1	Auto
0	Off
1	Conservative
2	Aggressive

zeroobjnodes (*integer)* Number of nodes to explore in the zero objective heuristic

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

(default = 0)