# LINDO/LINDOGlobal

**Lindo Systems, Inc.**

## Contents



## 1 Introduction

GAMS/LINDO finds guaranteed globally optimal solutions to general nonlinear problems with continuous and/or discrete variables. GAMS/LINDO supports most mathematical functions, including functions that are nonsmooth, such as abs(x) and or even discontinuous, such as floor(x). Nonlinear solvers employing methods like successive linear programming (SLP) or generalized reduced gradient (GRG) return a local optimal solution to an NLP problem. However, many practical nonlinear models are non-convex and have more than one local optimal solution. In some applications, the user may want to find a global optimal solution.

The LINDO global optimization procedure(GOP) employs branch-and-cut methods to break an NLP model down into a list of subproblems. Each subproblem is analyzed and either a) is shown to not have a feasible or optimal solution, or b) an optimal solution to the subproblem is found, e.g., because the subproblem is shown to be convex, or c) the subproblem is further split into two or more subproblems which are then placed on the list. Given appropriate tolerances, after a finite, though possibly large number of steps a solution provably global optimal to tolerances is returned. Traditional nonlinear solvers can get stuck at suboptimal, local solutions. This is no longer the case when using the global solver.

GAMS/LINDO can automatically linearize a number of nonlinear relationships, such as max(x,y), through the addition of constraints and integer variables, so the transformed linearized model is mathematically equivalent to the original nonlinear model. Keep in mind, however, that each of these strategies will require additional computation time. Thus, formulating models, so they are convex and contain a single extremum, is desirable. In order to decrease required computing power and time it is also possible to disable the global solver and use GAMS/LINDO like a regular nonlinear solver.

GAMS/LINDO has a multistart feature that restarts the standard (non-global) nonlinear solver from a number of intelligently generated points. This allows the solver to find a number of locally optimal points and report the best one found. This alternative can be used when global optimization is costly. A user adjustable parameter controls the maximum number of multistarts to be performed.

LINDO automatically detects problem type and uses an appropriate solver, e.g., if you submit an LP model to LINDO, it will be solved as an LP at LP speed, regardless of what you said in the ”solve using” statement. With the NLP parameter *NLP_QUADCHK* turned on, LINDO can detect hidden quadratic expressions and automatically recognize convex QCPs, as well as second-order cones (SOCP), like in Value-at-Risk models, allowing dramatically faster solution times via the barrier solver. When such models have integer variables, LINDO would use the barrier solver to solve all subproblems leading to significantly improved solution times when compared to the case with the standard NLP solver.

## 1.1 Licensing and software requirements

In order to use GAMS/LINDOGlobal, users need a GAMS/LINDOGlobal license. Additionally a GAMS/CONOPT license is required for solving nonlinear subproblems. The GAMS/LINDOGlobal license places upper limits on model size of 3,000 variables and 2,000 constraints.

To use GAMS/LINDO, no additional license is required. Also, there is no upper limit on the model size with GAMS/LINDO and, in addition, it allows to solve stochastic models (see section 7).

Neither the GAMS/LINDO not the GAMS/LINDOGlobal license does include the Barrier solver option. LINDO would be able to use the barrier solver when the user has a separate license for the GAMS/MOSEK barrier solver.

## 1.2 Running GAMS/LINDO

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

```
option xxx=lindo;
option xxx=lindoglobal;
```

where xxx is one of: LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP, or MINLP.

You can also find global optima to math programs with equilibrium or complementarity constraints, type MPEC, by using the GAMS/NLPEC translator in conjunction with LINDO. You use NLPEC to translate complementarities into standard mathematical statements, e.g. h*y = 0, and then use LINDO as the DNLP(Discontinuous Nonlinear) solver to solve the translated model. The following little GAMS model illustrates:

```
$TITLE simple mpec example
variable f, x1, x2, y1, y2; positive
variable y1; y2.lo = -1; y2.up = 1;

equations cost, g, h1, h2;

cost..  f =E= x1 + x2;
   g..  sqr(x1) + sqr(x2) =L= 1;
  h1..  x1 =G= y1 - y2 + 1;
  h2..  x2 + y2 =N= 0;

* declare h and y complementary
model example / cost, g, h1.y1, h2.y2 /;
```

```
option mpec=nlpec;
option dnlp=lindo;
solve example using mpec min f;
```

## 2 Supported nonlinear functions

GAMS/LINDO supports most nonlinear functions in global mode, including +, -, *, /, floor, modulo, sign, min, max, sqr, exp, power, ln, log, sqrt, abs, cos, sin, tan, cosh, sinh, tanh, arccos, arcsin, arctan and logic expressions AND, OR, NOT, and IF. Be aware that using highly nonconvex functions may lead to long solve times.

## 3 GAMS/LINDO output

The log output below is obtained for the NLP model mhw4d.gms from the GAMS model library using LINDOs global solver.

```
LINDO                24Nov11 23.8.0 WIN 30200.30202 VS8 x86/MS Windows

   LINDO Driver
   Lindo Systems Inc, www.lindo.com

Lindo API version 7.0.1.372 built on Nov  3 2011 21:49:01
Barrier Solver Version 6.0.0.114, Nonlinear Solver Version 3.15B
Platform Windows x86

Number of constraints:       3    le:       0, ge:       0, eq:       3, rn:
   0 (ne:0)
Number of variables  :       5    lb:       0, ub:       0, fr:       5, bx:
   0 (fx:0)
Number of nonzeroes  :       8    density=0.0053(%)

Nonlinear variables  :       5
Nonlinear constraints:       4
Nonlinear nonzeroes  :       5+5

Starting global optimization ...

Number of nonlinear functions/operators:  3
 EP_MULTIPLY  EP_POWER  EP_SQR

Starting GOP presolve ...
First Call Local Solver
Find local solution, objvalue =    27.871905

Pre-check unboundedness
Computing reduced bound...
Searching for a better solution...

Starting reformulation ...

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


 #NODEs  BOXES   LOWER BOUND     UPPER BOUND      RGAP   TIME(s)

     1       1  -3.167052e+022  +2.931083e-002  1.0e+000      0 (*N)
    19      17  -2.136461e+000  +2.931083e-002  1.0e+000      0 (*I)
    22      20  -1.848574e-001  +2.931083e-002  2.1e-001      0 (*I)
    23      21  +2.416053e-003  +2.931083e-002  2.7e-002      0 (*F)


Terminating global search ...

 Global optimum found
 Objective value              :       0.0293108307216
 Best Bound                   :      0.00241605257558
 Factors (ok,stb)             :                   522 (100.00,99.81)
 Simplex iterations           :                  2503
 Barrier iterations           :                     0
 Nonlinear iterations         :                   433
 Box iterations               :                    23
 Total number of boxes        :                    21
 Max. Depth                   :                     5
 First solution time (sec.)   :                     0
 Best solution time (sec.)    :                     0
 Total time (sec.)            :                     0
```

After determining the different kinds of nonlinear operators LINDO tries to linearize these within the presolving. When a feasible starting point is found the optimization starts and the log provides information about the progress. At the end it is reported if an optimum could be found and then the results as well as the used resources are summarized.

The following flags can be seen in the progress log:

| | |
|---|---|
| **(*FP)** | found a new MIP solution with feasibility pump |
| **(*SBB)** | found a new MIP solution in tree reorder |
| **(*SE)** | found a new MIP solution in simple enumeration |
| **(*AB)** | found a new MIP solution in advanced branching |
| **(*AH)** | found a new MIP solution with advanced heuristics |
| **(*C)** | found a new MIP solution after cuts added |
| **(*T)** | found a new MIP solution on the top |
| **(*SRH)** | found a new MIP solution in simple rounding heuristics |
| **(*SB)** | found a new MIP solution in strong branching |
| **(*K)** | found a new MIP solution in knapsack enumerator |
| **(*)** | found a new MIP solution normal branching |
| **(*?-)** | found a new MIP solution with advanced heuristics (level>10) |
| **(*N)** | found a new incumbent GOP solution |
| **(*I)** | stored a box with the incumbent solution into the GOP solution list |
| **(*F)** | determined the final GOP status |

# 4 Summary of GAMS/LINDO Options

GAMS/LINDO offers a diverse range of user-adjustable parameters to control the behavior of its solvers. While the default values of these parameters work best for most purposes, there may be cases the users prefer to work with different settings for a subset of the available parameters. This section gives a list of available GAMS/LINDO parameters, categorized by type, along with their brief descriptions. A more detailed description is given in the section that follows.

## 4.1 GAMS/LINDO Options File

In order to set GAMS/LINDO options, you need to set up an option file *lindo.opt* or *lindoglobal.opt* in your GAMS project directory. You must indicate in the model that you want to use the option file by inserting before the solve statement, the line:

```
<modelname>.optfile = 1;
```

where

```
<modelname>
```

is the name of the model referenced in the model statement. The option file is in plain text format containing a single GAMS/LINDO option per line. Each option identifier is followed by its target value with space or tab characters separating them. The lines starting with * character are treated as comments.

A sample option file *lindo.opt* looks like below

```
* Use(1) or Disable(0) global optimization for NLP/MINLP models
USEGOP          0

* Enable Multistart NLP solver
NLP_SOLVER      9

* Allow a maximum of 3 multistart attempts
MAXLOCALSEARCH      3

* Set an overall time limit of 200 secs.
SOLVER_TIMLMT       200
```

# 5 Summary of GAMS/LINDO Options

## 5.1 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 |
| SOLVER_PRE_ELIM_FILL | fill-in introduced by the eliminations during pre-solve |

## 5.2 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 explicit 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_IPMSOL | basis crossover flag for barrier solver |
| SOLVER_USECUTOFFVAL | flag for using cutoff value |
| SOLVER_CONCURRENT_OPTMODE | controls if simplex and interior-point optimizers will run concurrently |

## 5.3 IPM Options

| | |
|---|---|
| IPM_TOL_INFEAS | infeasibility tolerance |
| IPM_TOL_PATH | how close to follow the central path |
| IPM_TOL_PFEAS | primal feasibility tolerance |
| IPM_TOL_REL_STEP | relative step size to the boundary |
| IPM_TOL_PSAFE | controls the initial primal starting point |
| IPM_TOL_DFEAS | dual feasibility tolerance |
| IPM_TOL_DSAFE | controls the initial dual starting point |
| IPM_TOL_MU_RED | relative complementarity gap tolerance |
| IPM_BASIS_REL_TOL_S | maximum relative dual bound violation allowed in an optimal basic solution |
| IPM_BASIS_TOL_S | maximum absolute dual bound violation in an optimal basic solution |
| IPM_BASIS_TOL_X | maximum absolute primal bound violation allowed in an optimal basic solution |
| IPM_BI_LU_TOL_REL_PIV | relative pivot tolerance used in the LU factorization in the basis identification procedure |
| IPM_MAX_ITERATIONS | ipm iteration limit |
| IPM_OFF_COL_TRH | extent for detecting the offending columns in the Jacobian of the constraint matrix |
| IPM_NUM_THREADS | number of threads to run the interiorpoint optimizer on |

## 5.4 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 branch and bound 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 branch and bound 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 and 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 keeping 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 |
| MIP_HEU_MODE | heuristic used in MIP solver |
| MIP_FP_MODE | mode for the feasibility pump heuristic |
| MIP_FP_WEIGTH | weight of the objective function in the feasibility pump |
| MIP_FP_OPT_METHOD | optimization and reoptimization method for feasibility pump heuristic |
| MIP_FP_TIMLIM | time limit for feasibility pump heuristic |
| MIP_FP_ITRLIM | iteration limit for feasibility pump heuristic |
| MIP_SWITCHFAC_SIM_IPM_TIME | factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method |
| MIP_MAXNUM_MIP_SOL_STORAGE | maximum number of k-best solutions to store |
| MIP_PREHEU_LEVEL | heuristic level for the prerelax solver |
| MIP_PREHEU_TC_ITERLIM | iteration limit for the two change heuristic |
| MIP_PREHEU_DFE_VSTLIM | limit for the variable visit in depth first enumeration |

## 5.5 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 |
| NLP_FEASCHK | how to report results when solution satisfies tolerance of scaled but not original model |
| NLP_MSW_SOLIDX | index of the multistart solution to be loaded |
| NLP_ITERS_PER_LOGLINE | number of nonlinear iterations to elapse before next progress message |

| | |
|---|---|
| NLP_MAX_RETRY | maximum number refinement retries to purify the final NLP solution |
| NLP_MSW_NORM | norm to measure the distance between two points in multistart search |
| NLP_MSW_MAXREF | maximum number of reference points to generate trial points in multistart search |
| NLP_MSW_MAXPOP | maximum number of populations to generate in multistart search |
| NLP_MSW_MAXNOIMP | maximum number of consecutive populations to generate without any improvements |
| NLP_MSW_FILTMODE | filtering mode to exclude certain domains during sampling in multistart search |
| NLP_MSW_POXDIST_THRES | penalty function neighborhood threshold in multistart search |
| NLP_MSW_EUCDIST_THRES | euclidean distance threshold in multistart search |
| NLP_MSW_XNULRAD_FACTOR | initial solution neighborhood factor in multistart search |
| NLP_MSW_XKKTRAD_FACTOR | KKT solution neighborhood factor in multistart search |
| NLP_MAXLOCALSEARCH_TREE | maximum number of multistarts |

## 5.6 Global Options

| | |
|---|---|
| ABSOPTTOL | absolute optimality tolerance |
| RELOPTTOL | relative 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 |
| LIM_MODE | flag indicating which heuristic limit on sub-solver in GOP is based |
| ITRLIM | GOP iteration limit |
| ITRLIM_SIM | total simplex iteration limit summed over all branches in GOP |
| ITRLIM_IPM | total barrier iteration limit summed over all branches in GOP |
| ITRLIM_NLP | total nonlinear iteration limit summed over all branches in GOP |
| USEGOP | use global optimization |

## 5.7 SP Options

| | |
|---|---|
| REPORTEVSOL | solve and report the expected value solution |
| STOC_NSAMPLE_PER_STAGE | list of sample sizes per stage (starting at stage 2) |
| STOC_NSAMPLE_SPAR | common sample size per stochastic parameter |
| STOC_NSAMPLE_STAGE | common sample size per stage |

| | |
|---|---|
| STOC_RG_SEED | seed to initialize the random number generator |
| STOC_METHOD | stochastic optimization method to solve the model |
| STOC_REOPT | reoptimization method to solve the node-models |
| STOC_TOPOPT | optimization method to solve the root problem |
| STOC_ITER_LIM | iteration limit for stochastic solver |
| STOC_DETEQ_TYPE | type of deterministic equivalent |
| STOC_CALC_EVPI | flag to enable or disable calculation of EVPI |
| STOC_SAMP_CONT_ONLY | flag to restrict sampling to continuous stochastic parameters only or not |
| STOC_BUCKET_SIZE | bucket size in Benders decomposition |
| STOC_MAX_NUMSCENS | maximum number of scenarios before forcing automatic sampling |
| STOC_SHARE_BEGSTAGE | stage beyond which node-models are shared |
| STOC_NODELP_PRELEVEL | presolve level solving node-models |
| STOC_TIME_LIM | time limit for stochastic solver |
| STOC_RELOPTTOL | relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver |
| STOC_ABSOPTTOL | absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver |
| STOC_VARCONTROL_METHOD | sampling method for variance reduction |
| STOC_CORRELATION_TYPE | correlation type associated with correlation matrix |
| STOC_WSBAS | warm start basis for wait-see model |
| STOC_ALD_OUTER_ITER_LIM | outer loop iteration limit for ALD |
| STOC_ALD_INNER_ITER_LIM | inner loop iteration limit for ALD |
| STOC_ALD_DUAL_FEASTOL | dual feasibility tolerance for ALD |
| STOC_ALD_PRIMAL_FEASTOL | primal feasibility tolerance for ALD |
| STOC_ALD_DUAL_STEPLEN | dual step length for ALD |
| STOC_ALD_PRIMAL_STEPLEN | primal step length for ALD |
| CORE_ORDER_BY_STAGE | order nontemporal models or not |
| STOC_MAP_MPI2LP | flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements |
| STOC_AUTOAGGR | flag to enable or disable autoaggregation |
| STOC_BENCHMARK_SCEN | benchmark scenario to compare EVPI and EVMU against |
| STOC_INFBND | value to truncate infinite bounds at non-leaf nodes |
| STOC_ADD_MPI | flag to use add-instructions mode when building deteq |
| STOC_ELIM_FXVAR | flag to enable elimination of fixed variables from deteq MPI |
| STOC_SBD_OBJCUTVAL | RHS value of objective cut in SBD master problem |
| STOC_SBD_OBJCUTFLAG | flag to enable objective cut in SBD master problem |
| STOC_SBD_NUMCANDID | maximum number of candidate solutions to generate at SBD root |
| STOC_BIGM | big-M value for linearization and penalty functions |
| STOC_NAMEDATA_LEVEL | name data level |
| STOC_SBD_MAXCUTS | max cuts to generate for master problem |
| SAMP_NCM_METHOD | bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem |
| SAMP_NCM_CUTOBJ | objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem |
| SAMP_NCM_DSTORAGE | flag to enable or disable sparse mode in NCM computations |
| SAMP_CDSINC | correlation matrix diagonal shift increment |
| SAMP_SCALE | flag to enable scaling of raw sample data |
| SAMP_NCM_ITERLIM | iteration limit for NCM method |
| SAMP_NCM_OPTTOL | optimality tolerance for NCM method |
| STOC_DEQOPT | method to solve the DETEQ problem |
| SVR_LS_MONTECARLO | Sample variance reduction map to Lindo Montecarlo algorithm |
| SVR_LS_LATINSQUARE | Sample variance reduction map to Lindo Latin Square algorithm |
| SVR_LS_ANTITHETIC | Sample variance reduction map to Lindo Antithetic algorithm |

## 5.8 Link Options

| | |
|---|---|
| CHECKRANGE | calculate feasible range for variables |
| READPARAMS | read Lindo parameter file |
| WRITEDEMPI | write deterministic equivalent in MPI format |
| WRITEDEMPS | write deterministic equivalent in MPS format |
| WRITEMPI | write (S)MPI file of processed model |
| WRITEMPS | write (S)MPS file of processed model |

# 6 Detailed Descriptions of GAMS/LINDO 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 nonnegative integer, then it will be used as an upper bound on the number of iterations the solver will perform. If this value is -1, then no iteration limit will be used. The solution may be infeasible.

*(default = infinity)*

**SPLEX_PPRICING *(integer)***

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 explicit primal or dual form of the given LP problem will be solved.

*(default = 0)*

0 Solver decides

1 Explicit primal form

2 Explicit dual form

**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_IPMSOL (*integer*)**

This flag controls whether a basis crossover will be performed when solving LPs with the barrier solver. A value of 0 indicates that a crossover to a basic solution will be performed. If the value is 1, then the barrier solution will be left intact. For example, if alternate optima exist, the barrier method will return a solution that is, loosely speaking, the average of all alternate optima.

*(default = 0)*

0 Perform crossover to basis solution

1 Leave barrier solution intact

**SOLVER_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

**SOLVER_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 = 1000)*

**SOLVER_CONCURRENT_OPTMODE (*integer*)**

Controls if simplex and interior-point optimizers will run concurrently, 0 means no concurrent runs will be performed, 1 means both optimizers will run concurrently if at least two threads exist in system, 2 means both optimizers will run concurrently.

*(default = 0)*

0 no concurrent runs

1 run concurrently if at least 2 threads exist

2 run concurrently

**IPM_TOL_INFEAS (*real*)**

This is the tolerance to declare the model primal or dual infeasible using the interior-point optimizer. A smaller number means the optimizer gets more conservative about declaring the model infeasible.

*(default = 1e-8)*

**IPM_TOL_PATH (*real*)**

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. For numerically unstable problems it might help to increase this parameter.

*(default = 1e-8)*

**IPM_TOL_PFEAS (*real*)**

Primal feasibility tolerance used for linear and quadratic optimization problems.

*(default = 1e-8)*

**IPM_TOL_REL_STEP (*real*)**

Relative step size to the boundary for linear and quadratic optimization problems.

*(default = 0.9999)*

**IPM_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)*

**IPM_TOL_DFEAS (*real*)**

Dual feasibility tolerance used for linear and quadratic optimization problems.

*(default = 1e-8)*

**IPM_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 dual variables associated with constraint or variable bounds are very large, then it might be worthwhile to increase this value.

*(default = 1)*

**IPM_TOL_MU_RED (*real*)**

Relative complementarity gap tolerance.

*(default = 1e-16)*

**IPM_BASIS_REL_TOL_S (*real*)**

Maximum relative dual bound violation allowed in an optimal basic solution.

*(default = 1e-12)*

**IPM_BASIS_TOL_S (*real*)**

Maximum absolute dual bound violation in an optimal basic solution.

*(default = 1e-7)*

**IPM_BASIS_TOL_X (*real*)**

Maximum absolute primal bound violation allowed in an optimal basic solution.

*(default = 1e-7)*

**IPM_BI_LU_TOL_REL_PIV (*real*)**

Relative pivot tolerance used in the LU factorization in the basis identification procedure.

*(default = 1e-2)*

**IPM_MAX_ITERATIONS (*integer*)**

Controls the maximum number of iterations allowed in the interior-point optimizer.

*(default = 1000)*

**IPM_OFF_COL_TRH (*integer)***

Controls the extent for detecting the offending columns in the Jacobian of the constraint matrix. 0 means no offending columns will be detected. 1 means offending columns will be detected. In general, increasing the parameter value beyond the default value of 40 does not improve the result.

*(default = 40)*

**IPM_NUM_THREADS (*integer)***

Number of threads to run the interiorpoint optimizer on. This value should be less than or equal to the actual number of processors or cores on a multi-core system.

*(default = 1)*

**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

**NLP_FEASCHK (*integer*)**

This input parameter specifies how the NLP solver reports the results when an optimal or local-optimal solution satisfies the feasibililty tolerance (NLP_FEASTOL) of the scaled model but not the original (descaled) one.

*(default = 1)*

0 Perform no action accept the final solution

1 Declare the model status as FEASIBLE if maximum violation in the unscaled model is not higher than 10 times NLP_FEASTOL

2 Declare the model status as UNKNOWN if maximum violation in the unscaled model is higher than NLP_FEASTOL

**NLP_MSW_SOLIDX (*integer*)**

Index of the multistart solution to be loaded main solution structures.

*(default = 0)*

**NLP_ITERS_PER_LOGLINE (*integer*)**

Number of nonlinear iterations to elapse before next progress message.

*(default = 10)*

**NLP_MAX_RETRY (*integer*)**

Maximum number refinement retries to purify the final NLP solution.

*(default = 5)*

**NLP_MSW_NORM (*integer*)**

Norm to measure the distance between two points in multistart search.

*(default = 2)*

**NLP_MSW_MAXREF (*integer*)**

Maximum number of reference points in the solution space to generate trial points in multistart search.

*(default = -1)*

**NLP_MSW_MAXPOP (*integer*)**

Maximum number of populations to generate in multistart search.

*(default = -1)*

**NLP_MSW_MAXNOIMP (*integer*)**

Maximum number of consecutive populations to generate without any improvements.

*(default = -1)*

**NLP_MSW_FILTMODE (*integer*)**

Filtering mode to exclude certain domains during sampling in multistart search.

*(default = -1)*

-1 Solver decides

+1 Filter-out the points around known KKT or feasible points previously visited

+2 Filter-out the points whose p are in the vicinity of p(x)

+4 Filter-out the points in the vicinity of x where x are initial points of all previous local optimizations

+8 Filter-out the points whose p(.) values are below a dynamic threshold tolerance

**NLP_MSW_POXDIST_THRES (*real*)**

Penalty function neighborhood threshold in multistart search

*(default = 0.01)*

**NLP_MSW_EUCDIST_THRES (*real*)**

Euclidean distance threshold in multistart search

*(default = 0.001)*

**NLP_MSW_XNULRAD_FACTOR (*real*)**

Initial solution neighborhood factor in multistart search

*(default = 0.5)*

**NLP_MSW_XKKTRAD_FACTOR (*real*)**

KKT solution neighborhood factor in multistart search

*(default = 0.85)*

**NLP_MAXLOCALSEARCH_TREE (*integer*)**

Maximum number of multistarts (at tree nodes)

*(default = 1)*

**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 = 1022)*

| | |
|---|---|
| +2 | Simple pre-solving |
| +4 | Probing |
| +8 | Coefficient reduction |
| +16 | Elimination |
| +32 | Dual reductions |
| +64 | Use dual information |
| +128 | Binary row presolving |
| +256 | Row aggregation |
| +512 | Coef Probe Lifting |
| +1024 | Maximum pass |

**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 = GAMS OptCR)*

**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 = 22526)*

- +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 Branch and Bound 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-and-bound 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 (*real*)**

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 = infinity)*

**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 = GAMS OptCA)*

**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)*

0 Solver decides

1 Only use min pseudo cost

2 Only use max pseudo cost

3 Use quadratic score function and the pseudo cost weigth

4 Same as 3 without quadratic score

**MIP_USE_ENUM_HEU (*integer*)**

This specifies the frequency of enumeration heuristic.

*(default = 4)*

0 Off

1 Only at top (root) node without cuts

2 Both at top (root) and tree nodes without cuts

3 Same as 1 with cuts

4 Same as 2 with cuts

**MIP_PRELEVEL_TREE (*integer*)**

This controls the amount and type of MIP pre-solving at tree nodes.

*(default = 1214)*

+2 Simple pre-solving

+4 Probing

+8 Coefficient reduction

+16 Elimination

+32 Dual reductions

+64 Use dual information

+128 Binary row presolving

+256 Row aggregation

+512 Coef Probe Lifting

+1024 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)*

**MIP_HEU_MODE (*integer*)**

This controls the MIP heuristic mode.

*(default = 0)*

0 Solver decides when to stop the heuristic

1 Solver uses a pre-specified time limit to stop the heuristic.

2 Solver uses a pre-specified iteration limit to stop the heuristic

**MIP_FP_MODE (*integer*)**

Controls the mode for the feasibility pump heuristic.

*(default = 1)*

0 Off
1 Solver decides
2 On until the first solution
3 Try to get more than one solution

**MIP_FP_WEIGTH (*real)***

Controls the weight of the objective function in the feasibility pump.

*(default = 1)*

**MIP_FP_OPT_METHOD (*integer)***

This specifies optimization and reoptimization method for feasibility pump heuristic.

*(default = 0)*

0 Solver decides
1 Primal simplex
2 Dual simplex
3 Barrier

**MIP_FP_TIMLIM (*real)***

This is the time limit in seconds for feasibility pump heuristic. A value of -1 implies no time limit is imposed.

*(default = 1800)*

**MIP_FP_ITRLIM (*integer)***

This is the iteration limit in seconds for feasibility pump heuristic. A value of -1 means no iteration limit is imposed.

*(default = 500)*

**MIP_SWITCHFAC_SIM_IPM_TIME (*real)***

This specifies the (positive) factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method. A value of 1.0 means that no time limit is imposed.

*(default = -1)*

**MIP_MAXNUM_MIP_SOL_STORAGE (*integer)***

This specifies the maximum number of k-best solutions to store. Possible values are positive integers.

*(default = 1)*

**MIP_PREHEU_LEVEL (*integer)***

The heuristic level for the prerelax solver.

*(default = 0)*

0 Nothing
1 One-change
2 One-change and two-change
3 Depth first enumeration

**MIP_PREHEU_TC_ITERLIM (*integer)***

Iteration limit for the two change heuristic.

*(default = 30000000)*

**MIP_PREHEU_DFE_VSTLIM (*integer)***

Limit for the variable visit in depth first enumeration.

*(default = 200)*

**ABSOPTTOL (*real*)**

This value is the GOP absolute optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

*(default = GAMS OptCA)*

**RELOPTTOL (*real*)**

This value is the GOP relative optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

*(default = GAMS OptCR)*

**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 ABSOPTTOL at the global optimum. When its value is 1, the relative deviation of objective lower and upper bounds should be smaller than RELOPTTOL at the global optimum. 2 means either absolute or relative tolerance is satisfied at global optimum.

*(default = 2)*

**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-and-bound 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)*

0 Global search for a feasible solution (thus a feasibility certificate)

1 Global search for an optimal solution

2 Global search for an unboundedness certificate

**HEU_MODE (*integer)***

This specifies the heuristic used in the global solver to find a good solution. Typically, if a heuristic is used, this will put more efforts in searching for good solutions, and less in bound tightening.

*(default = 0)*

0 No heuristic is used

1 A simple heuristic is used

**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)*

**LIM_MODE (*integer)***

This is a flag indicating which heuristic limit on sub-solver in GOP is based.

*(default = 1)*

0 No limit

1 Time based limit

2 Iteration based limit

3 Both time and iteration based limit

**ITRLIM (*real)***

This is the total iteration limit (including simplex, barrier and nonlinear iteration) summed over branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

*(default = infinity)*

**ITRLIM_SIM (*real)***

This is the total simplex iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

*(default = -1)*

**ITRLIM_IPM (*real)***

This is the total barrier iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

*(default = -1)*

**ITRLIM_NLP (*real)***

This is the total nonlinear iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

*(default = -1)*

**REPORTEVSOL (*integer)***

*(default = 0)*

**STOC_NSAMPLE_PER_STAGE (*string)***

Comma separated list of sample sizes per stage. The sample size of stage 1 is assumed to be 1 so that this list starts with stage stage 2.

**STOC_NSAMPLE_SPAR (*integer)***

Common sample size per stochastic parameter. Possible values are positive integers.

*(default = -1)*

**STOC_NSAMPLE_STAGE (*integer)***

Common sample size per stage.

*(default = -1)*

**STOC_RG_SEED (*integer)***

Seed to initialize the random number generator. Possible values are positive integers.

*(default = 1031)*

**STOC_METHOD (*integer)***

Stochastic optimization method to solve the model.

*(default = -1)*

-1 Solve with the method chosen by the solver

0 Solve the deterministic equivalent (DETEQ)

1 Solve with the Nested Benders Decomposition (NBD) method

**STOC_REOPT (*integer)***

Reoptimization method to solve the node-models.

*(default = 0)*

0 Solver decides

1 Use primal method

2 Use dual simplex

3 Use barrier solver

4 Use NLP solver

**STOC_TOPOPT (*integer)***

Optimization method to solve the root problem.

*(default = 0)*

0 Solver decides

1 Use primal method

2 Use dual simplex

3 Use barrier solver

4 Use NLP solver

6 Use multi-start solver

7 Use global solver

**STOC_ITER_LIM (*integer)***

Iteration limit for stochastic solver. Possible values are positive integers or (-1) no limit.

*(default = infinity)*

**STOC_DETEQ_TYPE (*integer)***

Type of deterministic equivalent to be used by the solver. Implicit determinisitc equivalent is valid for linear and integer models only.

*(default = -1)*

-1 Solver decides

0 Implicit determinisitc equivalent

1 Explicit determinisitc equivalent

**STOC_CALC_EVPI (*integer*)**

Flag to enable/disable calculation of lower bounds on EVPI.

*(default = 1)*

0 disable
1 enable

**STOC_SAMP_CONT_ONLY (*integer*)**

Flag to restrict sampling to continuous stochastic parameters only or not.

*(default = 1)*

0 disable
1 enable

**STOC_BUCKET_SIZE (*integer*)**

Bucket size in Benders decomposition. Possible values are positive integers or (-1) for solver decides.

*(default = -1)*

**STOC_MAX_NUMSCENS (*integer*)**

Maximum number of scenarios before forcing automatic sampling. Possible values are positive integers.

*(default = 40000)*

**STOC_SHARE_BEGSTAGE (*integer*)**

Stage beyond which node-models share the same model structure. Possible values are positive integers less than or equal to number of stages in the model or (-1) for solver decides.

*(default = -1)*

**STOC_NODELP_PRELEVEL (*integer*)**

Presolve level solving node-models.

*(default = 0)*

+2 Simple pre-solving
+4 Probing
+8 Coefficient reduction
+16 Elimination
+32 Dual reductions
+64 Use dual information
+512 Maximum pass

**STOC_TIME_LIM (*real*)**

Time limit for stochastic solver. Possible values are nonnegative real numbers or -1 for solver decides.

*(default = GAMS ResLim)*

**STOC_RELOPTTOL (*real*)**

Relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. Possible values are reals in (0,1) interval.

*(default = GAMS OptCR)*

**STOC_ABSOPTTOL (*real*)**

Absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. . Possible values are reals in (0,1) interval.

*(default = GAMS OptCA)*

**STOC_VARCONTROL_METHOD (*integer*)**

Sampling method for variance reduction.

*(default = 1)*

0 Montecarlo sampling
1 Latinsquare sampling
2 Antithetic sampling

**STOC_CORRELATION_TYPE (*integer*)**

Correlation type associated with the correlation matrix.

*(default = 0)*

-1 Target correlation
0 Pearson correlation
1 Kendall correlation
2 Spearman correlation

**STOC_WSBAS (*integer*)**

Warm start basis for wait-see model .

*(default = -1)*

**STOC_ALD_OUTER_ITER_LIM (*integer*)**

Outer loop iteration limit for ALD.

*(default = 200)*

**STOC_ALD_INNER_ITER_LIM (*integer*)**

Inner loop iteration limit for ALD.

*(default = 1000)*

**STOC_ALD_DUAL_FEASTOL (*real*)**

Dual feasibility tolerance for ALD.

*(default = 1e-4)*

**STOC_ALD_PRIMAL_FEASTOL (*real*)**

Primal feasibility tolerance for ALD.

*(default = 1e-4)*

**STOC_ALD_DUAL_STEPLEN (*real*)**

Dual step length for ALD.

*(default = 0.9)*

**STOC_ALD_PRIMAL_STEPLEN (*real*)**

Primal step length for ALD.

*(default = 0.5)*

**CORE_ORDER_BY_STAGE (*integer*)**

Order nontemporal models or not.

*(default = 1)*

**STOC_MAP_MPI2LP (*integer*)**

Flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements.

*(default = 0)*

**STOC_AUTOAGGR (*integer*)**

Flag to enable or disable autoaggregation.

*(default = 1)*

**STOC_BENCHMARK_SCEN (*integer*)**

Benchmark scenario to compare EVPI and EVMU against.

*(default = -2)*

**STOC_INFBND (*real*)**

Value to truncate infinite bounds at nonleaf nodes.

*(default = 1e9)*

**STOC_ADD_MPI (*integer*)**

Flag to use add-instructions mode when building deteq.

*(default = 0)*

**STOC_ELIM_FXVAR (*integer*)**

Flag to enable elimination of fixed variables from deteq MPI.

*(default = 1)*

**STOC_SBD_OBJCUTVAL (*real*)**

RHS value of objective cut in SBD master problem.

*(default = 1e-30)*

**STOC_SBD_OBJCUTFLAG (*integer*)**

Flag to enable objective cut in SBD master problem.

*(default = 1)*

**STOC_SBD_NUMCANDID (*integer*)**

Maximum number of candidate solutions to generate at SBD root.

*(default = -1)*

**STOC_BIGM (*real*)**

Big-M value for linearization and penalty functions.

*(default = 1e7)*

**STOC_NAMEDATA_LEVEL (*integer*)**

Name data level.

*(default = 1)*

**STOC_SBD_MAXCUTS (*integer*)**

Max cuts to generate for master problem.

*(default = -1)*

**SAMP_NCM_METHOD (*integer*)**

Bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem.

*(default = 5)*

**SAMP_NCM_CUTOBJ (*real*)**

Objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem.

*(default = 1e-30)*

**SAMP_NCM_DSTORAGE (*integer*)**

Flag to enable/disable sparse mode in NCM computations.

*(default = -1)*

**SAMP_CDSINC (*real*)**

Correlation matrix diagonal shift increment.

*(default = 1e-6)*

**SAMP_SCALE (*integer*)**

Flag to enable scaling of raw sample data.

*(default = 0)*

**SAMP_NCM_ITERLIM (*integer*)**

Iteration limit for NCM method.

*(default = 100)*

**SAMP_NCM_OPTTOL (*real*)**

Optimality tolerance for NCM method.

*(default = 1e-7)*

**STOC_DEQOPT (*integer*)**

Optimization method to solve the DETEQ problem.

*(default = 0)*

0 Solver decides
1 Use primal method
2 Use dual simplex
3 Use barrier solver
4 Use NLP solver
6 Use multi-start solver
7 Use global solver

**SVR_LS_MONTECARLO (*string*)**

**SVR_LS_LATINSQUARE (*string*)**

**SVR_LS_ANTITHETIC (*string*)**

**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

**READPARAMS (*string*)**

**WRITEDEMPI (*string*)**

**WRITEDEMPS (*string*)**

**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.

**WRITEMPS (*string*)**

# 7 Stochastic Programming (SP) in GAMS/Lindo

GAMS/Lindo can also solve stochastic programming models. The syntax to set up an SP problem in GAMS is expalined in the chapter EMPSP. The options to control LINDOs stochastic solver are described in the subsection SP Options.