OQNLP

OQNLP

Introduction

Overview

Welcome to the TOMLAB /OQNLP User's Guide. The package includes the OQNLP solvers from Optimal Methods Inc. and an interface to The MathWorks' MATLAB. The TOMLAB /OQNLP package includes OQNLP, MSNLP and LSGRG2, while TOMLAB /MSNLP includes MSNLP and LSGRG2.

TOMLAB /OQNLP is a multistart heuristic algorithm designed to find global optima of smooth constrained nonlinear programs (NLPs) and mixed-integer nonlinear programs (MINLPs).

TOMLAB /MSNLP is also a multistart heuristic algorithm designed to find global optima of smooth constrained nonlinear programs (NLPs). The latest release support integers (MINLPs).

LSGRG2 can be called directly. The solver does not find global optima but only local solutions, which may be sufficient for many user cases. Up to 10,000 variables and constraints are supported

The multistart feature calls an NLP solver with a different set of initial values and return the feasible solutions as well as the optimal point. The starting points are calculated from scatter search algorithm, see http://www.opttek.com/ [1] for additional information. The user may also choose to use uniformly distributed initial values. Neither of the two options guarantee that a global optimum is obtained, however the likelihood is high.

The main advantage with OQNLP/MSNLP for smooth problems is that good local solutions are easily obtained, and that integer variables are handled.

Contents of this Manual

- #Overview provides a basic overview of the TOMLAB /OQNLP and TOMLAB /MSNLP solver packages.
- #Using the Matlab Interface provides an overview of the Matlab interfaces to OQNLP and MSNLP (LSGRG2).
- #Setting OQNLP Options describes how to set OQNLP and MSNLP (LSGRG2) solver options from Matlab.
- #OQNLP Solver Reference gives detailed information about the interface routines *oqnlpTL*, *msnlpTL* and *lsgrg2TL*.

More information

Please visit the following links for more information:

- http://tomopt.com/tomlab/products/oqnlp/ [2]
- http://www.opttek.com [1]

Prerequisites

In this manual we assume that the user is familiar with global optimization and nonlinear programming, setting up problems in TOMLAB (in particular constrained nonlinear (**con**) problems) and the Matlab language in general.

Using the Matlab Interface

The OQNLP and MSNLP (LSGRG2) solvers are accessed via the *tomRun* driver routine, which calls the appropriate interface routine (for example *oqnlpTL*). The solvers themselves are located in a set of MEX files: *oqnlp*, *msnlp* and *lsgrg2*.

Table: The interface routines.

Function	Description
oqnlpTL	The interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls the MEX file *oqnlp*
oqnlp_g	Interface routine called by OQNLP when an objective gradient is not given and integer variables are included.
oqnlp_dc	Interface routine called by OQNLP when a Jacobian is not given and integer variables are included.
oqnlp_gdc	Interface routine called by OQNLP when a gradient or Jacobian are not given using the *simAssign* format and integer variables are included.
msnlpTL	The interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls the MEX file *msnlp*
lsgrg2TL	The interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls the MEX file *lsgrg2*

Setting OQNLP Options

All OQNLP, MSNLP and LSGRG2 control parameters are possible to set from Matlab.

Setting options using the OQNLP.options structure

The parameters can be set as subfields in the *Prob.OQNLP.options* structure for TOMLAB /OQNLP and TOMLAB /MSNLP. If using the LSGRG2 solver, *Prob.LSGRG2.options* applies. The following example shows how to set a limit for the maximum number of iterations.

```
Prob = conAssign(...)                          % Setup problem, see help conAssign for more information
Prob.OQNLP.options.ITERATION_LIMIT  = 2000;      % Setting maximum number of iterations
```

The maximum number of iterations can also be done through the TOMLAB parameter *MaxIter*:

```
Prob.optParam.MaxIter = 200;
```

In the cases where a solver specific parameter has a corresponding TOMLAB general parameter, the latter is used only if the user has not given the solver specific parameter.

A complete description of the available parameters can be found in #OQNLP and MSNLP options.

OQNLP Solver Reference

A detailed description of the TOMLAB /OQNLP solver interface is given below. Also see the M-file help for *oqnlpTL* and *msnlpTL*.

oqnlpTL

Purpose

Solves global constrained nonlinear mixed-integer problems.

OQNLP solves problems of the form

$$\begin{array}{lccccc} \min\limits_{x} & f(x) & & & & \\ s/t & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array}$$

where $x, x_L, x_U \in R^n$, $A \in R^{m_1 \times n}$, $b_L, b_U \in R^{m_1}$and $c(x), c_L, c_U \in R^{m_2}$. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

Calling Syntax

```
Prob   = minlpAssign(...);
Result = tomRun('oqnlp',Prob,...)
```

Description of Inputs

Prob Problem description structure. The following fields are used:

Input	Description
A	Linear constraints coefficient matrix.
x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
c_L, c_U	Bounds on nonlinear constraints. For equality constraints (or fixed variables), set e.g. b_L(k) == b_U(k).
PriLevOpt	Print level in MEX interface.
LargeScale	Flag telling whether to treat the problem as sparse (1) or dense. If set to 1, the user should also provide a sparse 0-1 matrix in Prob.ConsPattern giving the nonzero pattern.
MIP	Structure with fields defining the integer properties of the problem. The following fields are used:
IntVars	Vector designating which variables are restricted to integer values. This field is interpreted differently depending on the length. If length(IntVars) = length(x), it is interpreted as a zero-one vector where all non-zero elements indicate integer values only for the corresponding variable. A length less than the problem dimension indicates that IntVars is a vector of indices for the integer variables, for example [1 2 3 6 7 12].
OQNLP	Structure with special fields for the OQNLP solver:
options	Structure array with options.
morereal	Number of extra REAL workspace locations. Set to <0 for problem dependent default strategy.
moreint	Number of extra INTEGER workspace locations. Set to <0 for problem de- pendent default strategy.

Description of Outputs

Result Structure with result from optimization. The following fields are set:

Output	Description
f_k	Function value at optimum.
g_k	Gradient of the function.
x_k	Solution vector.
x_0	Initial solution vector.
c_k	Nonlinear constraint residuals.
cJac	Nonlinear constraint gradients.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
cState	State of nonlinear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrange multipliers (for bounds + dual solution vector).

ExitFlag	Exit status from OQNLP (TOMLAB standard).
Inform	OQNLP information parameter. < 0 = Setup Error. 0 = Status not set. 1 = Optimal solution found. 2 = Fractional change in objective too small. 3 = All remedies failed. 4 = Too many iterations. 5 = Problem is unbounded. 6-10 = Problem infeasible. 11-38 = Runtime failure. 39 = Termination by user. 40 = Jacobian overflow. 41 = OPTQUEST Error. 42 = Time limit exceeded. 43 = Feasible solution found. Other = Unknown return code.
rc	Reduced costs. If ninf=0, last m == -v_k.
Iter	Number of iterations.
FuncEv	Number of function evaluations. *GradEv* Number of gradient evaluations. *ConstrEv* Number of constraint evaluations.
QP.B	Basis vector in TOMLAB QP standard.
Solver	Name of the solver (OQNLP).
SolverAlgorithm	Description of the solver.

msnlpTL

Purpose

Solves global constrained nonlinear problems.

MSNLP solves problems of the form

$$\min_x \quad f(x)$$

$$\begin{array}{lccccc} s/t & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array}$$

where $x, x_L, x_U \in R^n$, $A \in R^{m_1 \times n}$, $b_L, b_U \in R^{m_1}$and $c(x), c_L, c_U \in R^{m_2}$.

Calling Syntax

```
Prob = conAssign(...);
Result = tomRun('msnlp',Prob,...)
```

Description of Inputs

Prob Problem description structure. The following fields are used:

A	Linear constraints coefficient matrix.
x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
c_L, c_U	Bounds on nonlinear constraints. For equality constraints (or fixed variables), set e.g. b_L(k) == b_U(k).
PriLevOpt	Print level in MEX interface.
LargeScale	Flag telling whether to treat the problem as sparse (1) or dense. If set to 1, the user should also provide a sparse 0-1 matrix in Prob.ConsPattern giving the nonzero pattern.
MIP	Structure with fields defining the integer properties of the problem. The following fields are used:
OQNLP	Structure with special fields for the MSNLP solver:
options	Structure array with options.
morereal	Number of extra REAL workspace locations. Set to <0 for problem dependent default strategy.
moreint	Number of extra INTEGER workspace locations. Set to <0 for problem de- pendent default strategy.

Description of Outputs

Result Structure with result from optimization. The following fields are set:

Output	Description
f_k	Function value at optimum.
g_k	Gradient of the function.
x_k	Solution vector.
x_0	Initial solution vector.
c_k	Nonlinear constraint residuals.
cJac	Nonlinear constraint gradients.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
cState	State of nonlinear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrange multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from MSNLP (TOMLAB standard).

Inform	MSNLP information parameter. < 0 = Setup Error. 0 = Status not set. 1 = Optimal solution found. 2 = Fractional change in objective too small. 3 = All remedies failed. 4 = Too many iterations. 5 = Problem is unbounded. 6-10 = Problem infeasible. 11-38 = Runtime failure. 39 = Termination by user. 40 = Jacobian overflow. 41 = OPTQUEST Error. 42 = Time limit exceeded. 43 = Feasible solution found. Other = Unknown return code.
rc	Reduced costs. If ninf=0, last m == -v k.
Iter	Number of iterations.
FuncEv	Number of function evaluations.
GradEv	Number of gradient evaluations.
ConstrEv	Number of constraint evaluations.
QP.B	Basis vector in TOMLAB QP standard.
Solver	Name of the solver (MSNLP).
SolverAlgorithm	Description of the solver.

lsgrg2TL

Purpose

Solves constrained nonlinear problems.

LSGRG2 solves problems of the form

$$\min_x \quad f(x)$$

$$\begin{array}{lccccc} s/t & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array}$$

where $x, x_L, x_U \in R^n$, $A \in R^{m_1 \times n}$, $b_L, b_U \in R^{m_1}$and $c(x), c_L, c_U \in R^{m_2}$.

Calling Syntax

```
Prob = conAssign(...);
Result = tomRun('lsgrg2',Prob,...)
```

Description of Inputs

Prob Problem description structure. The following fields are used:

Input	Description
A	Linear constraints coefficient matrix.
x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
c_L, c_U	Bounds on nonlinear constraints. For equality constraints (or fixed variables), set e.g. b_L(k) == b_U(k).
PriLevOpt	Print level in optimizer and MEX interface. Set Prob.LSGRG2.options.IPR to set optimizer print level separately.
LargeScale	Flag telling whether to treat the problem as sparse (1) or dense. If set to 1, the user should also provide a sparse 0-1 matrix in Prob.ConsPattern giving the nonzero pattern.
MaxCPU	Maximum allowed time in seconds for the LSGRG2 run. It is also possible to set this through the Prob.LSGRG2.options.MAXTIME parameter, in which case Prob.MaxCPU is ignored. LSGRG2's default value for MAXTIME is 1000 seconds.
optParam.MaxIter	Maximum number of iterations. Default is 10000.
LSGRG2	Structure with special fields for the LSGRG2 solver:
options	Structure array with options. See the TOMLAB /OQNLP User's Guide for instructions and examples.
PrintFile	Name of file to receive the LSGRG2 iteration and results log. Independent of PriLevOpt.

Description of Outputs

Result Structure with result from optimization. The following fields are set:

Output	Description
f_k	Function value at optimum.
g_k	Gradient of the function.
x_k	Solution vector.
x_0	Initial solution vector.
c_k	Nonlinear constraint residuals.
cJac	Nonlinear constraint gradients.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equalit == 3;
cState	State of nonlinear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrange multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from LSGRG2 (TOMLAB standard).

Inform	LSGRG2 information parameter. < 0 = Setup Error. 0 = Status not set. 1 = Kuhn-Tucker conditions satisfied. 2 = Fractional change in objective too small. 3 = All remedies failed. 4 = Too many iterations. 5 = Problem is unbounded. 6-10 = Problem infeasible. 11-38 = Runtime failure. 39 = Termination by user. 40 = Jacobian overflow. 41 = Engine Error. 42 = Time limit exceeded. Other = Unknown return code.
rc	Reduced costs. If ninf=0, last m == -v k.
Iter	Number of iterations.
FuncEv	Number of function evaluations. *GradEv* Number of gradient evaluations. *ConstrEv* Number of constraint evaluations.
QP.B	Basis vector in TOMLAB QP standard.
Solver	Name of the solver (LSGRG2).
SolverAlgorithm	Description of the solver.

OQNLP and MSNLP options

The following table shows all the options that the user can set for the OQNLP and MSNLP solvers. The LSGRG2 solver options are in Table 9. Observe that parameters which mention OptQuest are only available when using OQNLP (The OptQuest engine is part of OQNLP). All parameters should be set in *Prob.OQNLP.options*

User options for the TOMLAB /OQNLP solvers. The following fields are used:

Option	Description	Default
BASIN_DECREASE_FACTOR	This value must be between 0 and 1. If DYNAMIC DISTANCE FILTER is set to 1, the MAXDIST value associated with any local solution is reduced by (1-BASIN_DECREASE_FACTOR) if MERIT WAITCYCLE consecutive trial points have distance from the solution less than MAXDIST.	0.2
BASIN_OVERLAP_FIX	A value of 1 turns on logic which checks the MAXDIST values of all pairs 1 of local solutions, and reduces any pair of MAXDIST values if their sum is greater then the distance between the 2 solutions. This ensures that the spherical models of their basins of attracting do not overlap. A value of 0 turns off this logic. Turning it off can reduce the number of NLP solver calls, but can also cause OQNLP to miss the global solution.	1
DISTANCE_FACTOR	If the distance between an OptQuest trial point and any local solution found previously is less then DISTANCE FACTOR * MAXDIST, the NLP solver is not started from that trial point. MAXDIST is the largest distance ever traveled to get to that local solution. Increasing DIS- TANCE FACTOR leads to fewer solver calls and risks finding a worse solution. Decreasing it leads to more solver calls and possibly a better solution.	1.0
DYNAMIC_DISTANCE_FILTER	A value of 1 turns on logic which reduces the value of MAXDIST 1 (described under the DISTANCE FILTER keyword) for a local solution if MERIT WAITCYCLE consecutive trial points have a distance from that solution less than MAXDIST. MAXDIST is multiplied by (1-BASIN REDUCTION FACTOR). A value of 0 turns off this logic. Turning it off can decrease the number of NLP solver calls, but can also lead to a worse final solution.	1

DYNAMIC_MERIT_FILTER	A value of 1 turns on logic which dynamically varies the pa- 1 rameters which increases the merit filter threshold, THRESH- OLD INCREASE FACTOR. If MERIT WAITCYCLE consecutive trial points have been rejected by the merit filter, this value is replaced by max(THRESHOLD INCREASE FACTOR, val), where val is the value of THRESHOLD INCREASE FACTOR which causes the merit filter to just accept the best of the previous MERIT WAITCYCLE trial points. A value of 0 turns off this logic. Turning it off can reduce NLP solver calls, but may lead to a worse final solution.	1
FEASIBILITY_TOLERANCE	This tolerance is used to check each point returned by an NLP solver for feasibility. If the largest absolute infeasibility at the point is larger than this tolerance, the point is classified infeasible. This test is made because points returned by NLP solvers may occasionally be infeasible despite feasible status codes. Some NLP solvers use internal scaling before testing for feasibility. The unscaled problem may be infeasible, while the scaled one is feasible. If this occurs, increasing this tolerance (to 1.e-2 or larger) often eliminates the problem.	1.e-4
FEASIBILITY_MODE	If this option is set to 1 the system focuses on finding a feasible point, 0 stopping after the first NLP solver call which finds such a point. If set to 0, the system tries to find the global optimum of the objective function subject to the constraints.	0
INFBND	This value (its negative) is given to OptQuest as the upper (lower) bound for any variable with no upper and lower bound. However, the original bounds are given to the local solver, so it can produce solutions not limited by this artificial bound. OptQuest must have finite upper and lower bounds for each variable. If INFBND (or any of the user-supplied bounds) is much larger than any component of the optimal solution, OptQuest will be less efficient because it is searching over a region that is much larger than needed. Hence the user is advised to try to provide realistic values for all upper and lower bounds. It is even more dangerous to make INFBND smaller then some component of a globally optimal solution, since OptQuest can never generate a trial point near that solution. It is possible, however, for the local solver to reach a global solution in this case, since the artificial bounds are not imposed on it.	1.e5
ITERATION_PRINT_FREQUENCY	If the OQNLP iteration log is written to the OQNLP log file, one line 20 of output is written every k'th OptQuest iteration, where k is the value given here.	20
ITERATION_LIMIT	Increasing this limit can allow OQNLP to find a better solution. Try it if your run using 1000 iterations doesn't take too long. Surprisingly, the best solution using, say 2000 iterations, may be found in the first 1000 iterations, and that solution may be better then the one found with an iteration limit of 1000. This is because OptQuest changes its search strategy depending on the iteration limit. Because of this, it is also possible that increasing the iteration limit will yield a worse solution, but this is rare. Decreasing this iteration limit usually leads to a worse solution, but also reduces run time.	1000
LOCALS_FILE	Specify a complete path and name for a file to which the objective value and values of all variables for all local solutions found will be written. For example, C:\temp\opt.out. There are 2 possible formats for this file, specified by the LOCALS FILE FORMAT option below. If there is no LOCALS FILE specified, the locals file will not be created. **WARNING**, no file will be created unless *Prob.OQNLP.PrintFile* is set.	
LOCALS_FILE_FORMAT	There are 2 possible values for this option. The REPORT entry creates the locals file in a format designed to be examined easily by eye, but processed less easily by a computer program or spreadsheet. The DATA1 entry creates a file with many records, each on a single line, each having the following format: < *indexof localoptimum* >< *objval* >< *varindex* >< *varindex* >.	DATA1
MAX_LOCALS	When the number of distinct local solutions found by OQNLP exceeds the value specified here, OQNLP will stop, returning the best solution found.	1000
MAX_SOLVER_CALLS	When the number of calls to the NLP solver exceeds the value specified here, OQNLP will stop, returning the best solution found.	1000

MAX_SOLVER_CALLS_NOIMPROVEMENT	The positive integer specified here will cause OQNLP to stop whenever the number of consecutive solver calls with a fractional improvement in the best objective value found less than 1.e-4 exceeds that values. In other words, if the value specified is 50, and there are more than 50 consecutive solver calls where the relative change in the best objective was less then 1.e-4 in all iterations, OQNLP will stop.	100
MAXTIME	When the execution time spent by OQNLP exceeds this number of sec- onds, OQNLP will stop and return the best solution found.	1000
MERIT_WAITCYCLE	This value must be a positive integer. If the merit filter is used, and there 20 are MERIT WAITCYCLE consecutive iterations where the merit filter logic causes the NLP solver not to be started, the merit filter threshold is increased by the factor THRESHOLD INCREASE FACTOR (see below). Increasing MERIT WAITCYCLE usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.	20
OPTQUEST_ONLY	If you think the NLP solver is taking too long and/or not working well, 0 choosing 1 will stop it from being called. This may occur if the problem is of type "DNLP", where one or more problem functions are discontin- uous or have discontinuous derivatives. If the problem has only discrete (integer) variables, choose 1, as there is nothing for the NLP solver to do (since it optimizes over the continuous variables when the integers are fixed, and there aren't any).	0
OQ_ALL_DISCRETE	This option sets the filter logic for problems with discrete variables. Setting this option to 1 has the potential to produce better solutions but also consumes more computing time.	0
PENALTY_FUNCTION	Results using either choice are usually similar, but choosing 1 can lead 1 to fewer iterations. The 1 value causes the exact penalty function to be used, while 0 uses OptQuest's internal penalty function, whose penalty term is the largest percentage violation of all violated constraints.	1
POINT GENERATION	*OPTQUEST* causes trial points to be generated by the OptQuest driver. This option is only available with TOMLAB /OQNLP and is the default for this option. *RANDOM* causes trial points to be generated by sampling each variable from a uniform distribution defined within its bounds. *SMARTRANDOM1* generates trial points by sampling each variable independently from either normal or triangular distributions. This is the default for TOMLAB /MSNLP.	See text
RANDOM_NUMBER_SEED	The options are: String 'RANDOM' = random seed computed from clock. String 'DEFAULT' = default seed value 1234. Any positive value = seed set to that value. Negative values = negated to positive. 0 = equivalent to 'DEFAULT'.	0
SAMPLING_DISTRIBUTION	This keyword is relevant only when POINT GENERATION is set to 0 SMARTRANDOM1. Then a value of 0 causes normal distributions to be used to generate trial points, while a value of 1 causes triangular distributions to be used.	0
SEARCH_PARAMETER	This parameter has a range between zero and one, and a default value of 0.5. Increasing it causes more trial points to be directed towards the boundary. This is advisable if you believe the optimal solution will be on the boundary or at a vertex, as is true for problems where a concave function is maximized subject to linear constraints.	0.5
SEARCH TYPE	This option controls the search strategy used by OptQuest. The three choices that are relevant for use within OQNLP are: **aggressive** This choice controls the population update of the OptQuest algorithm. It triggers a very aggressive update, which keeps the best of the points generated from the current population as the new population. The risk in this is that all points in the new population may cluster in a small portion of the search volume, and regions far from this volume will not be explored in the next cycle. **boundary** This option affects the trial points generated by OptQuest, directing them toward the boundary of the region defined by the linear constraints and variables bounds. The value of SEARCH PARAMETER discussed below controls the fraction of points that are directed towards the boundary. **crossover** This option affects how OptQuest trial points are generated from population points. It retains the linear combination operator, but adds a "crossover" operator, similar to those used in evolutionary or genetic algorithms, to create 2 additional trial points.	boundary

STAGE1_ITERATIONS	Specifies the total number of OptQuest iterations in stage 1 of the OQNLP algorithm, where no NLP solver calls are made. Increasing this sometimes leads to a better starting point for the first local solver call in stage 2, at the cost of delaying the call. Decreasing it can lead to more solver calls, but the first call occurs sooner.	200
STARTING_MULTIPLIER	Since no Lagrange multiplier values are available until the first solver call at the end of stage 1, this value is used for all multipliers if PENALTY_FUNCTION is set to 1.	1000
START_WITH_NLP_SOLVER	In the beginning, OQNLP passes the starting point defined in the TOMLAB model to the NLP subsolver. This will ensure that solutions gen- erated with OQNLP will be always as good as the one generated by the NLP subsolver alone. After this initial call the OQNLP algorithm con- tinues as usual. The initial call to the NLP subsolver starting from the point defined by the TOMLAB model can be suppressed by setting this parameter to 0.	1
THRESHOLD_INCREASE_FACTOR	This value must be nonnegative. If there are MERIT WAITCYCLE con- secutive OptQuest iterations where the merit filter logic causes the NLP solver not to be called, the merit threshold is increased by multiplying it by (1+THRESHOLD INCREASE FACTOR)	0.2
USE_DISTANCE_FILTER	Use 0 to turn off the distance filter, the logic which starts the NLP 1 solver at a trial point only if the (Euclidean) distance from that point to any local solution found thus far is greater than the distance threshold. Turning off the distance filter leads to more solver calls and more run time, and increases the chances of finding a global solution. Turn off both distance and merit filters to find (almost) all local solutions.	1
USE_LINEAR_CONSTRAINTS	This option applies only to problems which have linear constraints other 0 than simple bounds on the variables. Using 1 (all OptQuest trial points satisfy the linear constraints) often leads to fewer iterations and solver calls, but OptQuest has to solve an LP to project each trial point onto the linear constraints. For large problems (more than 100 variables), this can greatly increase run time, so the default value is off (0)	0
USE_MERIT_FILTER	Use 0 to turn off the merit filter, the logic which starts the NLP solver 1 at a trial point only if the penalty function value at that point is below the merit threshold. This will lead to more solver calls, but increase the chances of finding a global solution. Turn off both filters of you want to find (almost) all local solutions. This will cause the solver to be called at each stage 2 iteration.	1
WAITCYCLE	This value must be a positive integer. If the merit filter is used, and 20 there are WAITCYCLE consecutive iterations where the merit filter logic causes the NLP solver not to be started, the merit filter thresh- old is increased by the factor THRESHOLD INCREASE FACTOR (see above). Increasing WAITCYCLE usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.	20

LSGRG2 options

The following options are available for the LSGRG2 solver included with the TOMLAB /OQNLP /MSNLP package. The fields are set in *Prob.LSGRG2.options*. It is only possible to set these fields when calling LSGRG2 directly, i.e. they are ignored if calling OQNLP.

User options for the TOMLAB /OQNLP - LSGRG2 solver. The following fields are used:

Option	Description	Default
EPNEWT	A constraint is assumed to be binding if it is within this epsilon of one of its bounds.	1e-4
EPINIT	If it is desired to run the problem with EPNEWT initially set fairly large and then tightened at the end of the optimization then this is accomplished by assigning EPINIT the initial tolerance and EPNEWT the final one. Doing this often reduces run time.	1e-4
EPSTOP	If the fractional change in the objective is less than EPSTOP for NSTOP consecutive iterations, the program will stop. The program will also stop if Kuhn-Tucker optimality conditions are satisfied to within EPSTOP.	1e-4
EPSPIV	If, in constructing the basis inverse, the absolute value of a prospective pivot element is less than EPSPIV, the pivot will be rejected and another pivot element will be sought.	1e-6
PH1EPS	If nonzero, the phase 1 objective (normally the sum of infeasibilities, sinf) is aug- mented by a multiplier, say w, times the true objective, obj, so that the new phase 1 objective is sinf+w*obj. The multiplier, w, is selected so that, at the initial point, *w *obj = P H 1EP S *sinf* . In phase 1 the term *w *obj* should be small relative to sinf, so PH1EPS should be less than 1.0, with 0.1 a reasonable default value. The reason for setting PH1EPS positive is to cause phase 1 to pay a small amount of attention to the objective, so that any feasible point found will have a *good* objective value.	0.0
AIJTOL	Zero tolerance for Jacobian elements. Any derivative with absolute value less than this value will be set to zero.	1e-10
PIVPCT	When choosing pivots for each row, all columns with updated entries within PIVPCT of the maximum will be considered.	1e-1
PSTEP	This is the step size used in PARSHF and PARSHC for estimating partial derivatives of the functions with respect to the variables. This value is computed from the calculated machine precision.	1e-8
FUNPR	The function precision.	1e-8
CONDTL	When factoring the basis matrix, this is the largest condition number which is ac- ceptable. Blocks whose estimated condition number exceeds CONDTL are considered to be ill-conditioned, and an error message is issued to that effect. If you see these messages, either automatic or user scaling can improve the situation.	1e+8
EPBOUN	A variable is considered to be at its bound if it is within EPBOUN of the bound.	1e-6
EPDEG	If problem becomes degenerate, the variable bounds will be perturbed by this relative tolerance, and the problem resolved.	1e-4
MAXTIME	Time limit in seconds.	1000
NSTOP	If the fractional change in the objective is less than EPSTOP for NSTOP consecutive 3 iterations, the program will stop.	3
ITLIM	If subroutine newton takes ITLIM iterations without converging satisfactorily, the 10 iterations are stopped and corrective action is taken.	10
LIMSER	If the number of completed one dimensional searches equals LIMSER, optimization will terminate.	10000
INPRNT	If 1, turns on printing of initial problem structure, option settings, initial values, 0 bounds, and status of variables and rows.	0
OTPRNT	If 1, turns on printing of final values and status of rows and variables, Lagrange 0 multipliers, reduced gradients, and run statistics.	0
IPR	Suppress all output printing except initial and final reports. 1 = Print one line of output for each one dimensional search. 2 = Provide more detailed information on the progress of each one dimensional search. 3 = Expand the output to include the problem function values and variable values at each iteration as well as the separation of constraints into nonbinding and binding and variables into basic, superbasic and nonbasic. 4 = At each iteration the reduced gradient, the search direction and the tangent vector are printed. 5 = Provides details of the basis inversion process including the initial basis and its inverse. also displays the variable values and constraint errors for each newton iteration. 6 = This is the maximum level of print available and includes all of the above along with detailed progress of the basis construction phase, including the basis inverse at each pivot.	0
IPN#	If IPN# is greater than zero then IPR will be set to # after IPN# iterations. The 0 symbol # may be 4, 5, or 6. For example, setting IPN5 to 20 sets the print level IPR to 5 after 20 iterations.	0
IPER	If IPER is greater than zero then for every IPER th iteration, LSGRG2 output will 0 use the current value of IPR, otherwise use IPR=1.	0
IQUAD	Method for initial estimates of basic variables for each one dimensional search. 0 = Tangent vector and linear extrapolation will be used. 1 = Quadratic extrapolation will be used.	0

KDERIV	Method for obtaining partial derivatives. 0 = Forward difference approximation. 1 = Central difference approximation. 2 = User supplied subroutine parsh is used.	0
MODCG	MODCG and MAXHES (see below) control use of a conjugate gradient (cg) method. If the number of superbasic variables exceeds MAXHES, the cg method indicated by MODCG is used. Default value of *MODCG* = 6. To use a cg method at each iteration, set *MAXHES* = 0. 1 = Uses fletcher reeves formula. 2 = Uses polak ribiere formula. 3 = Uses perrys formula. 4 = Uses 1 step version of the DFP quasi-newton method. 5 = Uses 1 step version of the BFGS quasi-newton method. 6 = Uses limited memory version of the BFGS quasi-newton method. NOTE: if *MODC G* = 6, the optional parameters MEMCG and HSCALE (see below) are relevant.	0
MXTABU	The maximum length of the tabu list in chuzq.	25
IDEGLM	The maximum number of consecutive degenerate steps before the subroutine terminates.	25
ISCALE	Determines whether Jacobian scaling will be used at the initial point. 0 = Do not scale jacobian. 1 = Scale the jacobian.	0
ISCLAG	Rescale the Jacobian after every ISCLAG line searches are performed. 0 = No rescaling is performed. > 0 = Rescale after ISCLAG line searches.	0
MEMCG	If using *MODC G* = 6, MEMCG is the memory length of the limited memory cgmethod. Note: this method is only used if *MODCG* = 6.	3
IBVBLM	If a basic variable is in the basis at a bound for IBVBLM consecutive iterations, the LSGRG2 basis factorization routine does not try to replace it.	2
HRDBND	The maximum length of the tabu list in chuzq. 0 = This allows the user routines to be called with some variables outside their bounds. This can cause errors when routines contain functions like square roots, logs, etc. 1 = All variables will be within their bounds.	0
FIXPIV	The maximum length of the tabu list in chuzq. 0 = The pivot tolerance PIVPCT will be fixed at the specified level. 1 = The pivot tolerance PIVPCT will be adjusted dynamically.	0
GFEAS	Phase control. 0 = Both phase 1 and phase 2 are performed. 1 = Only a phase 1 is performed. The algorithm will terminate when a feasible point is found.	0
USEPH0	Phase control. 0 = No phase 0 is performed. Instead, a normal phase 1 is performed if the initial point is not feasible. 1 = A phase 0 is performed to attempt to find an initial feasible point via newtons method. This method is usually faster, but can occasionally cause LSGRG2 to fail, in which case it should be disabled.	1

LSGRG2 printing

The LSGRG2 printing logic is described below. Prob.PriLevOpt is called PriLev in the text.

PriLev controls the amount of output from LSGRG2, by setting the print level (IPR) to PriLev. PriLev also controls whether information shall be printed to the MATLAB screen or not. If PriLev is > 0, information from LSGRG2 is printed to the MATLAB screen.

Prob.LSGRG2.options.IPR is a solver specific option controlling the amount of output to be printed. It takes precedence to PriLev, i.e. an old IPR value set by PriLev is reset to the new Prob.LSGRG2.options.IPR value. IMPORTANT: PriLev remains PriLev, and it still controls any output to the MATLAB screen.

Prob.LSGRG2.PrintFile is the name of the output file. If set by the user and printing is turned on, the file will be created.

If *PriLev* <= 0 and printing is enabled through solver specific options and no Prob.LSGRG2.PrintFile name is given, a default file is opened.

If *PriLev* > 0 and printing is enabled in some way and no Prob.LSGRG2.PrintFile name is given, no default file will be opened, as the user may only want screen printing.

Options controlling printing are: Prob.LSGRG2.options.INPRNT, .OTPRNT, .IPR, .IPN\#, .IPER. In summary:

- Prob.PriLevOpt controls .IPR as long as .IPR is not given explicitly.
- No default print file is created if Prob.PriLevOpt is set. Although, if Prob.LSGRG2.PrintFile is set, a print file always is created, as long as there is something to print.

Algorithm

Multistart overview

A pseudo-code description of the MSNLP algorithm follows, in which SP(xt) denotes the starting point generator and xt is the candidate starting point produced. We refer to the local NLP solver as L(xs, xf), where xs is the starting point and xf the final point. The function UPDATE LOCALS(xs, xf, w) processes and stores solver output xf, using the starting point xs to compute the distance from xs to xf, and produces updated penalty weights, w.

MSNLP Algorithm

STAGE 1

x0 = user initial point

Call L(x0, xf)

Call UPDATE LOCALS(x0, xf,w)

FOR i = 1, n1 **DO** Call SP(xt(i)) Evaluate P(xt(i),w)

ENDDO

xt* = point yielding best value of P(xt(i),w) over all stage one points, (i = 1, 2, ..., n1). call L(xt*, xf)

Call UPDATE LOCALS(xt*, xf,w)

threshold = P(xt*,w)

STAGE 2

FOR i = 1, n2 **DO** Call SP(xt(i)) Evaluate P(xt(i),w)

Perform merit and distance filter tests: Call distance filter(xt(i), dstatus)

Call merit filter(xt(i), threshold, mstatus)

IF (dstatus and mstatus = 'accept') **THEN**

Call L(xt(i), xf)

Call UPDATE LOCALS(xt(i), xf, w)

ENDIF ENDDO

After an initial call to L at the user-provided initial point, x0, stage 1 of the algorithm performs n1 iterations in which SP(xt) is called, and the L1 exact penalty value P(xt,w) is calculated. The user can set n1 through the MSNLP options structure using the STAGE1 ITERATIONS keyword. The point with the smallest of these P values is chosen as the starting point for the next call to L, which begins stage 2. In this stage, n2 iterations are performed in which candidate starting points are generated and L is started at any one which passes the distance and merit filter tests.

The distance filter helps insure that the starting points for L are diverse, in the sense that they are not too close to any previously found local solution. Its goal is to prevent L from starting more than once within the basin of attraction of any local optimum. When a local solution is found, it is stored in a linked list, ordered by its objective value, as is the Euclidean distance between it and the starting point that led to it. If a local solution is located more than once, the maximum of these distances, maxdist, is updated and stored. For each trial point, t, if the distance between t and any local solution already found is less than DISTANCE FACTOR*maxdist, L is not started from the point, and we obtain the next trial solution from the generator.

This distance filter implicitly assumes that the attraction basins are spherical, with radii at least maxdist. The default value of DISTANCE FACTOR is 1.0, and it can be set to any positive value. As DISTANCE FACTOR approaches zero, the filtering effect vanishes, as would be appropriate if there were many closely spaced local solutions. As it becomes larger than 1, the filtering effect increases until eventually L is never started.

The merit filter helps insure that the starting points for L have high quality, by not starting from candidate points whose exact penalty function value P1 is greater than a threshold. This threshold is set initially to the P1 value of the best candidate point found in the first stage of the algorithm. If trial points are rejected by this test for more than WAITCYCLE consecutive iterations, the threshold is increased by the updating rule:

threshold < - threshold + THRESHOLD INCREASE FACTOR *(1.0 + *abs*(*threshold*))

where the default value of THRESHOLD INCREASE FACTOR is 0.2 and that for WAITCYCLE is 20. The additive 1.0 term is included so that threshold increases by at least THRESHOLD INCREASE FACTOR when its current value is near zero. When a trial point is accepted by the merit filter, threshold is decreased by setting it to the P1 value of that point.

The combined effect of these 2 filters is that L is started at only a few percent of the trial points, yet global optimal solutions are found for a very high percentage of the test problems. However, the chances of finding a global optimum are increased by increasing ITERATION LIMIT (which we recommend trying first) or by 'loosening' either or both filters, although this is rarely necessary if the dynamic filters and basin overlap fix are used, as they are by default. If the ratio of stage 2 iterations to solver calls is more than 20 using the current filter parameters, and computation times with the default filter parameters are reasonable, one can try loosening the filters. This is achieved for the merit filter either by decreasing WAITCYCLE or by increasing THRESHOLD INCREASE FACTOR (or doing both), and for the distance filter by decreasing DISTANCE FACTOR. Either or both filters may be turned off, by setting USE DISTANCE FILTER and/or USE MERIT FILTER to 0. Turning off both causes an NLP solver call at every stage 2 trial point. This is the best way to insure that all local optima are found, but it can take a long time.

Pure and smart random drivers

The 'pure' random (PR) driver generates uniformly distributed points within the hyper-rectangle S defined by the variable bounds. However, this rectangle is often very large, because users often set bounds to (-inf,+inf), (0,+inf), or to large positive and/or negative numbers, particularly in problems with many variables. This usually has little adverse impact on a good local solver, as long as the starting point is chosen well inside the bounds. But the PR generator will often generate starting points with very large absolute component values when some bounds are very large, and this sharply degrades solver performance. Thus we were motivated to develop random generators which control the likelihood of generating candidate points with large components, and intensify the search by focusing points into promising regions. We present two variants, one using normal, the other triangular distributions. Pseudo-code for this 'smart random' generator using normal distributions follows, where w is the set of penalty weights determined by the 'update locals' logic discussed above, after the first solver call at the user-specified initial point.

Smart Random Generator with Normal Distributions, SRN(xt)

IF (first call) **THEN**

Generate k1 (default 400) diverse points in S and evaluate the exact penalty function P(x,w) at each point. B=subset of S with k2 (default 10) best P values

FOR i = 1,nvars **DO**

xmax(i) = max of component i of points in B xmin(i)= min of component i of points in B mu(i) = (xmax(i) + xmin(i))/2

ratio(i) = (xmax(i) - xmin(i))/(1+buvar(i)-blvar(i))

sigfactor = 2.0

IF (ratio>0.7) sigfactor = f(ratio)

sigma(i) = (xmax(i) - xmin(i))/sigfactor

ENDDO ENDIF

FOR i = 1,nvars **DO**

Generate a normally distributed random variable rv(i) with mean mu(i) and standard deviation sigma(i) If rv(i) is between blvar(i) and buvar(i), xt(i) = rv(i)

If rv(i)<blvar(i), generate xt(i) uniformly between blvar(i) and xmin(i)

If rv(i)>buvar(i), generate xt(i) uniformly between xmax(i) and buvar(i)

ENDDO

Return xt

This SRN generator attempts to find a subset, B, of k2 'good' points, and generates most of its trial points xt, within the smallest rectangle containing B. It first generates a set of k1 diverse points within the bounds using a stratified random sampling procedure with frequency-based memory. For each variable x(i), this divides the interval \[blvar(i), buvar(i)\] into 4 equal segments, chooses a segment with probability inversely proportional to the frequency with which it has been chosen thus far, then generates a random point in this segment. We choose k2 of these points having the best P(x,w) penalty values, and use the smallest rectangle containing these, intersecting the ith axis at points [xmin(i), xmax(i)], to define n univariate normal distributions (driver SRN) or n univariate triangular distributions (driver SRT). The mean of the ith normal distribution, mu(i), is the midpoint of the interval \[xmin(i), xmax(i)\], and this point is also the mode of the ith triangular distribution, whose lower and upper limits are blvar(i) and buvar(i). The standard deviation of the ith normal distribution is selected as described below. The trial point xt is generated by sampling n times independently from these distributions. For the driver using normals, if the generated point lies within the bounds, it is accepted. Otherwise, we generate a uniformly distributed point between the violated bound and the start of the interval.

To determine the standard deviation of the normal distributions, we compute ratio, roughly the ratio of interval width to distance between bounds, where the factor 1.0 is included to avoid division by zero when the bounds are equal (fixed variables). If the interval width is small relative to the distance between bounds for variable i

(*ratio* = 0.7), then the standard deviation sigma(i) is half the interval width, so about 1/3 of the xt(i) values fall outside the interval, providing diversity when the interval does not contain an optimal value for x(i). If the bounds are large, then ratio should be small, say less than 0.1, so xt(i) values near the bounds are very unlikely. If ratio > 0.7, the function f sets sigfactor equal to 2.56 if ratio is between 0.7 and 0.8, increasing in steps to 6.2 if textitratio > 0.999. Thus if ratio is near 1.0, more than 99% of the values fall within the interval, and few have to be projected back within the bounds. The projecting back process avoids undesirable clustering of trial points at a bound, by generating points uniformly between the violated bound and the nearest edge of the interval \[xmin(i), xmax(i)\]. When the interval \[xmin(i), xmax(i)\] is sharply skewed toward one of the variable bounds and is much narrower than the distance between the bounds, a symmetric distribution like the normal, combined with our projection procedure, generates too many points between the interval and its nearest bound. A quick scan of the test results indicates that this happens rarely, but an asymmetric distribution like the triangular overcomes this difficulty, and needs no projection.

References

[1] http://www.opttek.com/

[2] http://tomopt.com/tomlab/products/oqnlp/

Article Sources and Contributors

OQNLP *Source*: http://tomwiki.com/index.php?oldid=2616 *Contributors*: Elias