TOMLAB QUICK START GUIDE

Marcus M. Edvall[1] and Anders Göran[2]

January 21, 2009

[1]Tomlab Optimization Inc., 855 Beech St #121, San Diego, CA, USA, medvall@tomopt.com.
[2]Tomlab Optimization AB, Västerås Technology Park, Trefasgatan 4, SE-721 30 Västerås, Sweden, anders@tomopt.com.

Contents

1 QuickGuide Overview

This guide will introduce you to the very basics associated with solving problems using the TOMLAB Optimization Environment. After this session you will have solved several different problems with a variety of solvers. The solvers you can use will depend on what you are licensed for. The total time required for these exercises has been estimated to about 45 minutes.

Useful commands when using this guide:

```
help *Assign (* is lp, qp, con and more)
help tomRun
SolverList('qp');
help snoptTL
help mipSolve
help minlpBBTL
help cplexTL
Prob - view the Problem structure used by the solvers
Result - view the Result structure
```

2 LP Problem

The general formulation in TOMLAB for a linear programming problem is:

$$\begin{aligned} \min_x \quad & f(x) = c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{1}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. Equality constraints are defined by setting the lower bound to the upper bound.

Example problem:

$$\begin{aligned} \min_{x_1,x_2} \quad & f(x_1, x_2) = -7x_1 - 5x_2 \\ s/t \quad & \begin{array}{ccc} x_1 + 2x_2 & \leq & 6 \\ 4x_1 + x_2 & \leq & 12 \\ x_1, x_2 & \geq & 0 \end{array} \end{aligned} \tag{2}$$

The following file defines this problem in TOMLAB.

File: tomlab/quickguide/lpQG.m

Open the file for viewing, and execute lpQG in Matlab.

```
% lpQG is a small example problem for defining and solving
% linear programming problems using the TOMLAB format.

Name  = 'lpQG';     % Problem name, not required.
c     = [-7 -5]';   % Coefficients in linear objective function
A     = [ 1  2
          4  1 ];   % Matrix defining linear constraints
b_U   = [ 6 12 ]';  % Upper bounds on the linear inequalities
x_L   = [ 0  0 ]';  % Lower bounds on x

% x_min and x_max are only needed if doing plots
x_min = [ 0  0 ]';
x_max = [10 10 ]';

% b_L, x_U and x_0 have default values and need not be defined.
% It is possible to call lpAssign with empty [] arguments instead
b_L   = [-inf -inf]';
x_U   = [];
x_0   = [];

% Assign routine for defining an LP problem. This allows the user
% to try any solver, including general nonlinear solvers.
Prob = lpAssign(c, A, b_L, b_U, x_L, x_U, x_0, Name,...
```

```
                  [], [], [], x_min, x_max, [], []);

% Calling driver routine tomRun to run the solver.
% The 1 sets the print level after optimization.

% Result.x_k contains the optimal decision variables.
% Result.f_k is the optimal value.

Result = tomRun('pdco', Prob, 1);

%Result = tomRun('lpSimplex', Prob, 1);
%Result = tomRun('minos', Prob, 1);
%Result = tomRun('snopt', Prob, 1);
%Result = tomRun('conopt', Prob, 1);
%Result = tomRun('knitro', Prob, 1);
%Result = tomRun('cplex', Prob, 1);
%Result = tomRun('xpress-mp', Prob, 1);
```

3 MILP Problem

The general formulation in TOMLAB for a mixed-integer linear programming problem is:

$$\begin{array}{lll} \min\limits_{x} & f(x) = c^T x & \\ & & \\ s/t & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U,\ x_j \in \mathbb{N} \quad \forall j \in \mathrm{I} \end{array} \end{array} \tag{3}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. The variables $x \in I$, the index subset of $1, ..., n$ are restricted to be integers. Equality constraints are defined by setting the lower bound equal to the upper bound, i.e. for constraint i: $b_L(i) = b_U(i)$.

Mixed-integer linear problems are defined in the same manner as linear problems. However, the user can give a wider range of inputs to the assign routine and solvers. See 'help mipAssign' for more information. In TOMLAB integers can be identified by a 0-1 vector.

The following example illustrates how to solve a MILP problem using the TOMLAB format.

File: tomlab/quickguide/mipQG.m

Open the file for viewing, and execute mipQG in Matlab.

```
% mipQG is a small example problem for defining and solving
% mixed-integer linear programming problems using the TOMLAB format.

Name='Weingartner 1 - 2/28 0-1 knapsack';
% Problem formulated as a minimum problem
A = [ 45      0     85     150     65     95     30      0    170  0 ...
      40     25     20       0      0     25      0      0     25  0 ...
      165     0     85       0      0      0      0    100  ; ...
      30     20    125       5     80     25     35     73     12  15 ...
      15     40      5      10     10     12     10      9      0  20 ...
      60     40     50      36     49     40     19    150];
b_U = [600;600];  % 2 knapsack capacities
c   = [1898  440  22507   270  14148  3100  4650  30800   615  4975 ...
       1160 4225    510 11880    479   440   490    330   110   560 ...
       24355 2885  11748  4550    750  3720  1950  10500]'; % 28 weights

% Make problem on standard form for mipSolve
[m,n]   = size(A);
c       = -c;           % Change sign to make a minimum problem
x_L     = zeros(n,1);
x_U     = ones(n,1);
x_0     = zeros(n,1);

fprintf('Knapsack problem. Variables %d. Knapsacks %d\n',n,m);

IntVars = [1:n];  % All original variables should be integer
x_min   = x_L; x_max  = x_U; f_Low  = -1E7; % f_Low <= f_optimal must hold
```

```
b_L     = -inf*ones(2,1);
f_opt   = -141278;

nProblem  = []; % Problem number not used
fIP       = []; % Do not use any prior knowledge
xIP       = []; % Do not use any prior knowledge
setupFile = []; % Just define the Prob structure, not any permanent setup file
x_opt     = []; % The optimal integer solution is not known
VarWeight = []; % No variable priorities, largest fractional part will be used
KNAPSACK  = 1;  % Run with the knapsack heuristic

% Assign routine for defining a MIP problem.
Prob      = mipAssign(c, A, b_L, b_U, x_L, x_U, x_0, Name, setupFile, ...
                      nProblem, IntVars, VarWeight, KNAPSACK, fIP, xIP, ...
                      f_Low, x_min, x_max, f_opt, x_opt);

Prob.optParam.IterPrint = 0; % Set to 1 to see iterations.
Prob.Solver.Alg = 2;   % Depth First, then Breadth search

% Calling driver routine tomRun to run the solver.
% The 1 sets the print level after optimization.

Result = tomRun('mipSolve', Prob, 1);
%Result = tomRun('cplex', Prob, 1);
%Result = tomRun('xpress-mp', Prob, 1);
%Result = tomRun('miqpBB', Prob, 1);
%Result = tomRun('minlpBB', Prob, 1);
```

4 QP Problem

The general formulation in TOMLAB for a quadratic programming problem is:

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2} x^T F x + c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{4}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. Equality constraints are defined by setting the lower bound equal to the upper bound, i.e. for constraint i: $b_L(i) = b_U(i)$. Fixed variables are handled the same way.

Example problem:

$$\begin{aligned} \min_x \quad & f(x) = 4x_1^2 + 1x_1x_2 + 4x_2^2 + 3x_1 - 4x_2 \\ s/t \quad & \begin{array}{l} x_1 + x_2 \leq 5 \\ x_1 - x_2 = 0 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{array} \end{aligned} \tag{5}$$

The following file defines this problem in TOMLAB.

File: tomlab/quickguide/qpQG.m

Open the file for viewing, and execute qpQG in Matlab.

```
% qpQG is a small example problem for defining and solving
% quadratic programming problems using the TOMLAB format.

Name  = 'QP Example';
F     = [ 8   1        % Matrix F in 1/2 * x' * F * x + c' * x
          1   8 ];
c     = [ 3  -4 ]';    % Vector c in 1/2 * x' * F * x + c' * x
A     = [ 1   1        % Constraint matrix
          1  -1 ];
b_L   = [-inf  0  ]';  % Lower bounds on the linear constraints
b_U   = [  5   0  ]';  % Upper bounds on the linear constraints
x_L   = [  0   0  ]';  % Lower bounds on the variables
x_U   = [ inf inf ]';  % Upper bounds on the variables
x_0   = [  0   1  ]';  % Starting point
x_min = [-1 -1 ];      % Plot region lower bound parameters
x_max = [ 6  6 ];      % Plot region upper bound parameters

% Assign routine for defining a QP problem.
Prob = qpAssign(F, c, A, b_L, b_U, x_L, x_U, x_0, Name,...
       [], [], [], x_min, x_max);
```

```
% Calling driver routine tomRun to run the solver.
% The 1 sets the print level after optimization.

Result = tomRun('qpSolve', Prob, 1);
%Result = tomRun('snopt', Prob, 1);
%Result = tomRun('sqopt', Prob, 1);
%Result = tomRun('cplex', Prob, 1);
%Result = tomRun('knitro', Prob, 1);
%Result = tomRun('conopt', Prob, 1);
```

5 MIQP Problem

The general formulation in TOMLAB for a mixed-integer quadratic programming problem is:

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2} x^T F x + c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U, \; x_j \in \mathbb{N} \quad \forall j \in \mathrm{I} \end{array} \end{aligned} \tag{6}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. The variables $x \in I$, the index subset of $1, ..., n$ are restricted to be integers. Equality constraints are defined by setting the lower bound equal to the upper bound, i.e. for constraint i: $b_L(i) = b_U(i)$.

The following file illustrates how to solve a MIQP problem in TOMLAB.

File: tomlab/quickguide/miqpQG.m

Open the file for viewing, and execute miqpQG in Matlab.

```
% miqpQG is a small example problem for defining and solving
% mixed-integer quadratic programming problems using the TOMLAB format.

c    = [-6 0]';
Name = 'XP Ref Manual MIQP';
F    = [4 -2;-2 4];
A    = [1 1];
b_L  = -Inf;
b_U  = 1.9;
x_L  = [0 0]';
x_U  = [Inf Inf]';

% Defining first variable as an integer
IntVars   = 1;

% Assign routine for defining a MIQP problem.
Prob = miqpAssign(F, c, A, b_L, b_U, x_L, x_U, [], ...
           IntVars, [], [], [], Name, [], []);

% Calling driver routine tomRun to run the solver.
% The 1 sets the print level after optimization.

Result = tomRun('cplex', Prob, 1);
%Result = tomRun('oqnlp', Prob, 1);
%Result = tomRun('miqpBB', Prob, 1);
%Result = tomRun('xpress-mp', Prob, 1);
%Result = tomRun('minlpBB', Prob, 1);
```

6 MIQQ Problem

The general formulation in TOMLAB for a mixed-integer quadratic programming problem with quadratic constraints is:

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x & & & & \\ s/t & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \\ & & & x^T Q^{(i)} x + a^{(i)T} x & \leq & r_U^{(i)}, \quad i = 1, \ldots, n_{qc} \\ & & & x_i \ \text{ integer} & & i \in I \end{array} \tag{7}$$

where $c, x, x_L, x_U, a^{(i)} \in \mathbb{R}^n$, $F, Q^{(i)} \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m \times n}$ and $b_L, b_U \in \mathbb{R}^m$. $r_U^{(i)}$ is a scalar. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

The following file illustrates how to solve a MIQQ problem in TOMLAB.

File: tomlab/quickguide/miqqQG.m

Open the file for viewing, and execute miqqQG in Matlab.

```
% miqqQG is a small example problem for defining and solving
% mixed-integer quadratic programming problems with quadratic constraints
% using the TOMLAB format.

Name = 'MIQQ Test Problem 1';
f_Low = -1E5;
x_opt = [];
f_opt = [];
IntVars = logical([0 0 1]); % 3rd variable is integer valued

F   = [2 0 0;0 2 0;0 0 2];
A   = [1 2 -1;1 -1 1];
b_L = [4 -2]';
b_U = b_L;
c   = zeros(3,1);

x_0 = [0 0 0]';
x_L = [-10 -10 -10]';
x_U = [10 10 10]';
x_min = [0 0 -1]';
x_max = [2 2 1]';

% Adding quadratic constraints
clear qc
qc(1).Q = speye(3,3);
qc(1).a = zeros(3,1);
qc(1).r_U = 3;

qc(2).Q = speye(3,3);
```

```
qc(2).a = zeros(3,1);
qc(2).r_U = 5;

Prob = miqqAssign(F, c, A, b_L, b_U, x_L, x_U, x_0, qc,...
                  IntVars, [], [], [],...
                  Name, [], [],...
                  x_min, x_max, f_opt, x_opt);

Result = tomRun('cplex', Prob, 1);
% Result = tomRun('minlpBB', Prob, 1);
```

7 NLP Problem

TOMLAB requires that general nonlinear problems are defined in Matlab m-files. The function to be optimized must always be supplied. It is recommended that the user supply as many analytical functions as possible. There are six methods available for numerical differentiation and also two for automatic.

The constrained nonlinear programming problem is defined as:

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{8}$$

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

Example problem:

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) = \alpha \left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 & & & & \\ & & & & & \\ & -10 & \leq & x_1 & \leq & 2 \\ s/t & -10 & \leq & x_2 & \leq & 2 \\ & \alpha = 100 & & & & \end{array} \tag{9}$$

The following files define the problem in TOMLAB.

File: tomlab/quickguide/rbb_f.m, rbb_g.m, rbb_H.m, rbb_c.m, rbb_dc.m, rbb_d2c.m

```
f:   Function value
g:   Gradient vector
H:   Hessian matrix
c:   Nonlinear constraint vector
dc:  Nonlinear constraint gradient matrix
d2c: The second part of the Hessian to the Lagrangian function for the nonlinear constraints.
```

The following file illustrates how to solve this NLP (CON) problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/nlpQG.m

Open the file for viewing, and execute nlpQG in Matlab.

```
% nlpQG is a small example problem for defining and solving
% nonlinear programming problems using the TOMLAB format.

Name = 'RBB Problem';
x_0 = [-1.2 1]';     % Starting values for the optimization.
x_L = [-10;-10];     % Lower bounds for x.
x_U = [2;2];         % Upper bounds for x.
fLowBnd = 0;         % Lower bound on function.
```

```
c_L = -1000;         % Lower bound on nonlinear constraints.
c_U = 0;             % Upper bound on nonlinear constraints.

Prob = conAssign('rbbQG_f', 'rbbQG_g', 'rbbQG_H', [], x_L, x_U, Name, x_0,...
    [], fLowBnd, [], [], [], 'rbbQG_c', 'rbbQG_dc', 'rbbQG_d2c', [], c_L, c_U);

Prob.Warning = 0;    % Turning off warnings.

Result = tomRun('ucSolve', Prob, 1);  % Ignores constraints.

% Result = tomRun('conopt', Prob, 1);
% Result = tomRun('snopt', Prob, 1);
```

8 LPCON Problem

When solving a problem with a linear objective and nonlinear constraints there is no need to explicitly code the gradient or Hessian. TOMLAB automatically supplies these if *lpconAssign* is used.

The linear constrained nonlinear programming problem is defined as:

$$\begin{aligned} \min_x \quad & f(x) = d^T x \\ s/t \quad & \begin{array}{rcccl} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \end{array} \end{aligned} \tag{10}$$

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

The following files define an example problem in TOMLAB.

File: tomlab/quickguide/lpconQG.m, lpconQG_c.m, lpconQG_dc.m, lpcon_d2c.m

```
c:   Nonlinear constraint vector
dc:  Nonlinear constraint gradient matrix
d2c: The second part of the Hessian to the Lagrangian function for the nonlinear constraints.
```

The following file illustrates how to solve this LPCON problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/lpconQG.m

Open the file for viewing, and execute lpconQG in Matlab.

```
% lpconQG is a small example problem for defining and solving linear
% nonlinearly constrained programming problems using the TOMLAB format.

Name = 'Linear constrained problem';
A = [1  0 -1  0
     0  1  0 -1
     0  0  1 -1];
b_L = [1 1 0]';
b_U = [];
c_L = -1;
c_U = -1;

x_0 = [1 1 1 1]';      % Starting value (not used)

x_L = [-Inf -Inf -Inf 1]'; % Lower bounds for x
x_U = 100*ones(4,1);     % Upper bounds for x

% Objective f = 3*x(1)+2*x(2), assign as linear vector

d = [3 2 0 0]';
```

```
Prob = lpconAssign(d, x_L, x_U, Name, x_0, A, b_L, b_U,...
                   'lpconQG_c', 'lpconQG_dc', 'lpconQG_d2c', [], c_L, c_U);

% Run a global solver for 10000 function evaluations.
Result = tomRun('glcFast', Prob, 1);

% Assign new starting point.
Prob.x_0 = Result.x_k(:,1);

% Run SNOPT as a local solver.

Result2 = tomRun('snopt', Prob, 1);

% Try KNITRO as a local solver, ALG = 3
% Prob.KNITRO.options.ALG = 3;
% Result3 = tomRun('knitro', Prob, 1);
```

9 QPCON Problem

When solving a problem with a quadratic objective and nonlinear constraints TOMLAB automatically supplies objective derivatives (gradient and Hessian) if *qpconAssign* is used.

The quadratic constrained nonlinear programming problem is defined as:

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x^T F x + d^T x \\ s/t \quad & \begin{array}{rcccl} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \end{array} \end{aligned} \tag{11}$$

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

The following files define and solve an example problem in TOMLAB.

File: tomlab/quickguide/qpconQG.m, qpconQG_c.m, qpconQG_dc.m, qpcon_d2c.m

```
c:   Nonlinear constraint vector
dc:  Nonlinear constraint gradient matrix
d2c: The second part of the Hessian to the Lagrangian function for the nonlinear constraints.
```

The following file illustrates how to solve this QPCON problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/qpconQG.m

Open the file for viewing, and execute qpconQG in Matlab.

```
% qpconQG is a small example problem for defining and solving quadratic
% nonlinearly constrained programming problems using the TOMLAB format.

Name  = 'QP constrained problem';
x_0   = ones(10,1);
x_L   = [];
x_U   = [];
A     = ones(8,10);
for i = 1:8
    A(i,i) = 1/2;
end
b_L   = ones(8,1);
b_U   = ones(8,1);
c_L   = 4;
c_U   = 4;

% Objective f = -x'*x + sum(x), assign as quadratic/linear matrix/vector

F     = -2*speye(10);
d     = ones(10,1);
```

```
Prob = qpconAssign(F, d, x_L, x_U, Name, x_0, A, b_L, b_U,...
    'qpconQG_c', 'qpconQG_dc', 'qpconQG_d2c', [], c_L, c_U);

% Run SNOPT as a local solver
Result = tomRun('snopt', Prob, 1);
% Result2 = tomRun('minos', Prob, 1);
% Result3 = tomRun('npsol', Prob, 1);
% Result4 = tomRun('knitro', Prob, 1);
```

10 MINLP Problem

The **mixed-integer nonlinear programming** (**minlp**) problem is defined as

$$\begin{array}{lllllll} \min\limits_{x} & f(x) & & & & & \\ & & & & & & \\ & -\infty < & x_L & \leq & x & \leq & x_U & < \infty \\ s/t & & b_L & \leq & Ax & \leq & b_U & \\ & & c_L & \leq & c(x) & \leq & c_U, \quad x_j \in \mathbb{N} \quad \forall j \in \mathrm{I}, \end{array} \tag{12}$$

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

The following files define a problem in TOMLAB.

File: tomlab/quickguide/minlpQG_f.m, minlpQG_g.m, minlpQG_H.m, minlpQG_c.m, minlpQG_dc.m, minlpQG_d2c.m

```
f:   Function value
g:   Gradient vector
H:   Hessian matrix
c:   Nonlinear constraint vector
dc:  Nonlinear constraint gradient matrix
d2c: The second part of the Hessian to the Lagrangian function for the nonlinear constraints.
```

The following file illustrates how to solve a MINLP problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/minlpQG.m

Open the file for viewing, and execute minlpQG in Matlab.

```
% minlpQG is a small example problem for defining and solving
% mixed-integer nonlinear programming problems using the TOMLAB format.

Name='minlp1Demo - Kocis/Grossman.';

IntVars   = logical([ 0 0 1 1 1 ]); % Integer variables: x(3)-x(5)
VarWeight = [ ];           % No priorities given

% There are divisions and square roots involving x(2), so we must
% have a small but positive value for the lower bound on x(2).
BIG = 1E8;

x_L = [ 0   1/BIG   0 0 0 ]';    % Lower bounds on x
x_U = [ BIG  BIG    1 1 1 ]';   % Upper bounds on x

% Three linear constraints
A = [1 0     1  0 0 ; ...
     0 1.333 0  1 0 ; ...
```

```
     0 0    -1 -1 1 ];

b_L = [];               % No lower bounds
b_U = [1.6 ; 3 ; 0];    % Upper bounds

c_L = [1.25;3];         % Two nonlinear constraints
c_U = c_L;              % c_L==c_U implies equality

x_0 = ones(5,1);        % Initial value

x_opt = [1.12,1.31,0,1,1]';  % One optimum known
f_opt = 7.6672;              % Value f(x_opt)

x_min = [-1 -1 0 0 0];  % Used for plotting, lower bounds
x_max = [ 1  1 1 1 1]; % Used for plotting, upper bounds

HessPattern = spalloc(5,5,0);  % All elements in Hessian are zero.
ConsPattern = [ 1 0 1 0 0; ...  % Sparsity pattern of nonlinear
        0 1 0 1 0 ];    % constraint gradient

fIP = [];        % An upper bound on the IP value wanted. Makes it possible
xIP = [];        % to cut branches. xIP: the x value giving fIP

% Generate the problem structure using the TOMLAB Quick format
Prob = minlpAssign('minlpQG_f', 'minlpQG_g', 'minlpQG_H', HessPattern, ...
                  x_L, x_U, Name, x_0, ...
                  IntVars, VarWeight, fIP, xIP, ...
                  A, b_L, b_U, 'minlpQG_c', 'minlpQG_dc', 'minlpQG_d2c', ...
          ConsPattern, c_L, c_U, ...
                  x_min, x_max, f_opt, x_opt);
Prob.DUNDEE.optPar(20) = 1;
Prob.P = 1;   % Needed in minlpQG_xxx files

% Get default TOMLAB solver for your current license, for "minlp" problems
% Solver = GetSolver('minlp');

% Call driver routine tomRun, 3rd argument > 0 implies call to PrintResult

Result  = tomRun('minlpBB',Prob,2);
```

11 LLS Problem

The **linear least squares** (**lls**) problem is defined as

$$\begin{array}{lll} \min\limits_{x} & f(x) = \frac{1}{2}||Cx - d|| & \\ & & \\ s/t & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} & \end{array} \tag{13}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $d \in \mathbb{R}^M$, $C \in \mathbb{R}^{M \times n}$, $A \in \mathbb{R}^{m_1 \times n}$ and $b_L, b_U \in \mathbb{R}^{m_1}$.

The following file defines and solves a problem in TOMLAB.

File: tomlab/quickguide/llsQG.m

Open the file for viewing, and execute llsQG in Matlab.

```
% llsQG is a small example problem for defining and solving
% linear least squares using the TOMLAB format.
Name='LSSOL test example';           % Problem name, not required.
n = 9;
x_L = [-2 -2 -inf, -2*ones(1,6)]';   % Lower bounds on x
x_U = 2*ones(9,1);                   % Upper bounds on x

% Matrix defining linear constraints
A   = [ ones(1,8) 4; 1:4,-2,1 1 1 1; 1 -1 1 -1, ones(1,5)];
b_L = [2    -inf -4]';    % Lower bounds on the linear inequalities
b_U = [inf    -2 -2]';    % Upper bounds on the linear inequalities

% Vector m x 1 with observations in objective ||Cx -y(t)||
y = ones(10,1);

% Matrix m x n in objective ||Cx -y(t)||
C = [ ones(1,n); 1 2 1 1 1 1 2 0 0; 1 1 3 1 1 1 -1 -1 -3; ...
      1 1 1 4 1 1 1 1 1;1 1 1 3 1 1 1 1 1;1 1 2 1 1 0 0 0 -1; ...
      1 1 1 1 0 1 1 1 1;1 1 1 0 1 1 1 1 1;1 1 0 1 1 1 2 2 3; ...
      1 0 1 1 1 1 0 2 2];

% Starting point.
x_0 = 1./[1:n]';

% x_min and x_max are only needed if doing plots.
x_min = -ones(n,1);
x_max =  ones(n,1);

% x_opt estimate.
x_opt = [2 1.57195927 -1.44540327 -0.03700275 0.54668583 0.17512363 ...
          -1.65670447 -0.39474418  0.31002899];
f_opt = 0.1390587318; % Estimated optimum.
```

```
% See 'help llsAssign' for more information.
Prob = llsAssign(C, y, x_L, x_U, Name, x_0, ...
                            [], [], [], ...
                            A, b_L, b_U, ...
                            x_min, x_max, f_opt, x_opt);

Result = tomRun('clsSolve', Prob, 1);
%Result = tomRun('nlssol', Prob, 1);
%Result = tomRun('snopt', Prob, 1);
%Result = tomRun('lssol', Prob, 1);
```

12 MILLS Problem

The **mixed-integer linear least squares** (**mills**) problem is defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}||Cx - d|| \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{14}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $d \in \mathbb{R}^M$, $C \in \mathbb{R}^{M \times n}$, $A \in \mathbb{R}^{m_1 \times n}$ and $b_L, b_U \in \mathbb{R}^{m_1}$. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

The following file defines and solves a problem in TOMLAB.

File: tomlab/quickguide/millsQG.m

Open the file for viewing, and execute millsQG in Matlab.

```
% millsQG is a small example problem for defining and solving
% mixed-integer linear least squares using the TOMLAB format.
Name='MILLS test example';           % Problem name, not required.
n = 10;
x_L = zeros(n,1);                    % Lower bounds on x
x_U = 20*ones(n,1);                  % Upper bounds on x

% Matrix defining linear constraints
A   = [ ones(1,n) ; 1:10; -2 1 -1 1 -1 1 -1 1 -1 1];
b_L = [35  -inf  29]';    % Lower bounds on the linear inequalities
b_U = [inf  120 210]';    % Upper bounds on the linear inequalities

% Vector m x 1 with observations in objective ||Cx -y(t)||
y = 2.5*ones(10,1);

% Matrix m x n in objective ||Cx -y(t)||
C = [ ones(1,n); 1 2 1 1 1 1 2 0 0 0; 1 1 3 1 1 1 -1 -1 -3 1; ...
   1 1 1 4 1 1 1 1 1 0;1 1 1 3 1 1 1 1 1 0;1 1 2 1 1 0 0 0 -1 1; ...
   1 1 1 1 0 1 1 1 1 0;1 1 1 0 1 1 1 1 1 1;1 1 0 1 1 1 2 2 3 0; ...
   1 0 1 1 1 1 0 2 2 1];

% Starting point.
x_0 = 1./[1:n]';

% x_min and x_max are only needed if doing plots.
x_min = -ones(n,1);
x_max =  ones(n,1);

% See 'help llsAssign' for more information.
Prob = llsAssign(C, y, x_L, x_U, Name, x_0, ...
   [], [], [], ...
   A, b_L, b_U);
```

```
IntVars = logical([ones(n-2,1);zeros(2,1)]);

Prob = lls2qp(Prob, IntVars);

Result = tomRun('cplex', Prob, 1);
% Result = tomRun('minlpBB', Prob, 1);
```

13 NLLS Problem

The **constrained nonlinear least squares** (**cls**) problem is defined as

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) = \frac{1}{2} r(x)^T r(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U, \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{15}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $r(x) \in \mathbb{R}^M$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$. The following file defines and solves a problem in TOMLAB.

The following files define a problem in TOMLAB.

File: tomlab/quickguide/nllsQG_r.m, nllsQG_J.m

```
r:   Residual vector
J:   Jacobian matrix
```

The following file illustrates how to solve an NLLS problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/nllsQG.m

Open the file for viewing, and execute nllsQG in Matlab.

```
% nllsQG is a small example problem for defining and solving
% nonlinear least squares using the TOMLAB format.
Name='Gisela';

t = [0.25; 0.5; 0.75; 1; 1.5; 2; 3; 4; 6; 8; 12; 24; 32; 48; 54; 72; 80;...
     96; 121; 144; 168; 192; 216; 246; 276; 324; 348; 386];
y = [30.5; 44; 43; 41.5; 38.6; 38.6; 39; 41; 37; 37; 24; 32; 29; 23; 21;...
     19; 17; 14; 9.5; 8.5; 7; 6; 6; 4.5; 3.6; 3; 2.2; 1.6];

x_0 = [6.8729, 0.0108, 0.1248]';

% See help clsAssign for more information.
Prob = clsAssign('nllsQG_r', 'nllsQG_J', [], [], [], Name, x_0, ...
                 y, t);

% Parameter which is passed to r and J routines.
Prob.uP = 5;

Result = tomRun('clsSolve', Prob, 1);
%Result = tomRun('nlssol', Prob, 1);

% Any nonlinear solver can be used. TOMLAB automatically
% uses gateway routines for problem mapping.
```

```
%Result = tomRun('filterSQP', Prob, 1);
%Result = tomRun('knitro', Prob, 1);
%Result = tomRun('conopt', Prob, 1);
%Result = tomRun('snopt', Prob, 1);
%Result = tomRun('npsol', Prob, 1);
%Result = tomRun('minos', Prob, 1);
%Result = tomRun('oqnlp', Prob, 1);
```

14 GLB Problem

The **unconstrained global optimization** (**glb**) problem is defined as

$$\begin{aligned} \min_x \quad & f(x) \\ s/t \quad & -\infty < \quad x_L \quad \leq \quad x \quad \leq \quad x_U \quad < \infty \end{aligned} \tag{16}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$.

The following files define a problem in TOMLAB.

File: tomlab/quickguide/glbQG_f.m

```
f:   Function
```

The following file illustrates how to solve an unconstrained global optimization problem in TOMLAB. Also view the m-file specified above for more information.

File: tomlab/quickguide/glbQG.m

Open the file for viewing, and execute glbQG in Matlab.

```
% glbQG is a small example problem for defining and solving
% unconstrained global programming problems using the TOMLAB format.

Name  = 'Shekel 5';
x_L   = [ 0  0  0  0]';  % Lower bounds for x.
x_U   = [10 10 10 10]';  % Upper bounds for x.
x_0   = [-3.0144 -2.4794 -3.1584 -3.1790]; % Most often not used.
x_opt = [];
f_opt = -10.1531996790582;
f_Low = -20;             % Lower bound on function.
x_min = [ 0  0  0  0]; % For plotting
x_max = [10 10 10 10]; % For plotting

Prob  = glcAssign('glbQG_f', x_L, x_U, Name, [], [], [], ...
                  [], [], [], x_0, ...
                  [], [], [], [], ...
                  f_Low, x_min, x_max, f_opt, x_opt);

Prob.optParam.MaxFunc = 1500;

Result1 = tomRun('glbFast', Prob, 1);  % Global solver
Result2 = tomRun('conSolve', Prob, 2); % Local solver, starting from Prob.x_0
% Also possible to use a mixed-integer global solver
Result = tomRun('glcDirect', Prob, 1);

% Result = tomRun('glbDirect', Prob, 1);
% Result = tomRun('glcDirect', Prob, 1);
```

```
% Result = tomRun('glbSolve', Prob, 1);
% Result = tomRun('glcSolve', Prob, 1);
% Result = tomRun('glcFast', Prob, 1);
% Result = tomRun('lgo', Prob, 1);
% Result = tomRun('oqnlp', Prob, 1);
```

15 GLC Problem

The **global mixed-integer nonlinear programming** (**glc**) problem is defined as

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) & & & & \\ & & & & & \\ & -\infty < & x_L & \leq \quad x \quad \leq & x_U & < \infty \\ s/t & & b_L & \leq \quad Ax \quad \leq & b_U & \\ & & c_L & \leq \quad c(x) \quad \leq & c_U, \quad x_j \in \mathbb{N} & \forall j \in \mathrm{I}, \end{array} \tag{17}$$

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

The following files define a problem in TOMLAB.

File: tomlab/quickguide/glcQG_f.m, glcQG_c.m

```
f:   Function
c:   Constraints
```

The following file illustrates how to solve a constrained global optimization problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/glcQG.m

Open the file for viewing, and execute glcQG in Matlab.

```
% glcQG is a small example problem for defining and solving
% constrained global programming problems using the TOMLAB format.

Name = 'Hock-Schittkowski 59';
u = [75.196    3.8112    0.0020567  1.0345E-5  6.8306    0.030234   1.28134E-3 ...
     2.266E-7  0.25645   0.0034604  1.3514E-5  28.106    5.2375E-6  6.3E-8     ...
     7E-10     3.405E-4  1.6638E-6  2.8673     3.5256E-5];

x_L = [0 0]';     % Lower bounds for x.
x_U = [75 65]';   % Upper bounds for x.
b_L = []; b_U = []; A = []; % Linear constraints
c_L = [0 0 0];    % Lower bounds for nonlinear constraints.
c_U = [];         % Upper bounds for nonlinear constraints.
x_opt = [13.55010424 51.66018129]; % Optimum vector
f_opt = -7.804226324;              % Optimum
x_min = x_L;      % For plotting
x_max = x_U;      % For plotting
x_0 = [90 10]';   % If running local solver

Prob = glcAssign('glcQG_f', x_L, x_U, Name, A, b_L, b_U, ...
                 'glcQG_c', c_L, c_U, x_0, ...
                 [], [], [], [], ...
                 [], x_min, x_max, f_opt, x_opt);
```

```
Prob.user.u = u;
Prob.optParam.MaxFunc = 1500;

Result = tomRun('glcFast', Prob, 1);
%Result = tomRun('glcSolve', Prob, 1);
%Result = tomRun('lgo', Prob, 1);
%Result = tomRun('oqnlp', Prob, 1);
```

16 SDP Problem

The **linear semi-definite programming problem with linear matrix inequalities** (**sdp**) is defined as

$$\begin{array}{lll} \min\limits_{x} & f(x) = c^T x & \\ & & \\ s/t & x_L \leq x \leq x_U & \\ & b_L \leq Ax \leq b_U & \\ & Q_0^i + \sum\limits_{k=1}^{n} Q_k^i x_k \preccurlyeq 0, \qquad i = 1, \ldots, m. & \end{array} \tag{18}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_l \times n}$, $b_L, b_U \in \mathbb{R}^{m_l}$ and Q_k^i are symmetric matrices of similar dimensions in each constraint i. If there are several LMI constraints, each may have it's own dimension.

The following file defines and solves a problem in TOMLAB.

File: tomlab/quickguide/sdpQG.m

Open the file for viewing, and execute sdpQG in Matlab.

This problem appears to be infeasible.

```
% sdpQG is a small example problem for defining and solving
% semi definite programming problems with linear matrix
% inequalities using the TOMLAB format.

Name = 'sdp.ps example 2';

% Objective function
c = [1 2 3]';

% Two linear constraints
A =   [ 0 0 1 ; 5 6 0 ];
b_L = [-Inf; -Inf];
b_U = [ 3  ; -3 ];

x_L = -1000*ones(3,1);
x_U =  1000*ones(3,1);

% Two linear matrix inequality constraints. It is OK to give only
% the upper triangular part.
SDP = [];
% First constraint
SDP(1).Q{1} = [2 -1 0 ; 0 2 0 ; 0 0 2];
SDP(1).Q{2} = [2 0 -1 ; 0 2 0 ; 0 0 2];
SDP(1).Qidx = [1; 3];

% Second constraint
SDP(2).Q{1} = diag( [0  1] );
SDP(2).Q{2} = diag( [1 -1] );
```

```
SDP(2).Q{3} = diag( [3 -3] );
SDP(2).Qidx = [0; 1; 2];

x_0 = [];

Prob = sdpAssign(c, SDP, A, b_L, b_U, x_L, x_U, x_0, Name);

Result = tomRun('pensdp', Prob, 1);
```

17 BMI Problem

The **linear semi-definite programming problem with bilinear matrix inequalities** (**bmi**) is defined similarly to SDP 18 but with the matrix inequality

$$Q_0^i + \sum_{k=1}^{n} Q_k^i x_k + \sum_{k=1}^{n} \sum_{l=1}^{n} x_k x_l K_{kl}^i \preccurlyeq 0 \tag{19}$$

The following file defines and solves a problem in TOMLAB.

File: tomlab/quickguide/bmiQG.m

Open the file for viewing, and execute bmiQG in Matlab.

```
% bmiQG is a small example problem for defining and solving
% semi definite programming problems with bilinear matrix
% inequalities using the TOMLAB format.

Name='bmi.ps example 3';
A   = [];
b_U = [];
b_L = [];

c   = [ 0 0 1 ];  % cost vector

% One matrix constraint, set linear part first
SDP = [];

% The constant matrix is stored as
% SDP(i).Q{j} when SDP(i).Qidx(j) == 0
SDP(1).Q{1}  = [-10 -0.5 -2 ;-0.5 4.5 0 ;-2 0 0 ];

SDP(1).Q{2} = [ 9 0.5 0       ;  0.5  0  -3 ;  0  -3 -1 ];
SDP(1).Q{3} = [-1.8 -0.1 -0.4 ; -0.1 1.2 -1 ; -0.4 -1 0 ];

% Sparse is fine, too. Eventually, all the matrices are
% converted to sparse format.
SDP(1).Q{4} = -speye(3);

SDP(1).Qidx = [0; 1; 2; 3];

% Now bilinear part

% K_12 of constraint 1 (of 1) is nonzero, so set in SDP(i).K{1}.
SDP(1).K{1} = [0 0 2 ; 0 -5.5 3 ; 2 3 0 ];
SDP(1).Kidx = [1 2];
n   = length(c);
```

```
x_L = [-5 ; -3 ; -Inf];
x_U = [ 2 ;  7 ;  Inf];
x_0 = [ 0 ;  0 ;  0  ];

f_Low = [];

Prob = bmiAssign([], c, SDP, A, b_L, b_U, x_L, x_U, x_0,...
                 Name, f_Low);

Result = tomRun('penbmi', Prob, 1);
```

18 MINIMAX Problem

The **constrained minimax** (**mima**) problem is defined as

$$\begin{array}{lllllll} \min\limits_{x} & \max r(x) & & & & & \\ \text{subject to} & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{20}$$

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

The following files define a problem in TOMLAB.

File: tomlab/quickguide/mimaQG_r.m, mimaQG_J.m

```
r:   Residual vector
J:   Jacobian matrix
```

The following file illustrates how to solve a minimax problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/minimaxQG.m

Open the file for viewing, and execute minimaxQG in Matlab.

```
% minimaxQG is a small example problem for defining and solving
% minimax programming problems using the TOMLAB format.

Name = 'Madsen-Tinglett 2';
x_0  = [1;1];        % Initial value
x_L  = [-10;-10];    % Lower bounds on x
x_U  = [10;10];      % Upper bounds on x

% Solve the problem min max |r_i(x)|, where i = 1,2,3
% Solve the problem by eliminating abs, doubling the residuals, reverse sign
% i.e. min max [r_1; r_2; r_3; -r_1; -r_2; -r_3];
y = [-1.5; -2.25; -2.625]; % The data values
y = [y ; -y];           % Eliminate abs, double the residuals, reverse sign
t = [];                 % No time vector used

%
% Add the linear constraint -x(1) + x(2) + 2 >= 0
% Write the constraint as x(1) - x(2) <= 2

% The A matrix could be specified dense or sparse
% A   = sparse([1 -1]);

A   = [1 -1];
b_L = -inf;
b_U = 2;
```

```
% Generate the problem structure using the Tomlab Quick format
%
% The part in the residuals dependent on x are defined in mima_r.m
% The Jacobian is defined in mima_J.m

Prob = clsAssign('mimaQG_r', 'mimaQG_J', [], x_L, x_U, Name, x_0, y, t, ...
                 [],[],[],[],A,b_L,b_U);

% Set the optimal values into the structure (for nice result presenting)

Prob.x_opt=[2.3660254038, 0.3660254038];
% Optimal residuals:
r_opt = [0; 0.2009618943;0.375];

% Compute optimal function value:
Prob.f_opt=max(abs(r_opt));

% Use the standard method in infSolve
Prob.InfType = 1;

% Get the default solver
% Solver = GetSolver('con',1,0);

Prob.SolverInf = 'conSolve';

% One may set other solvers:
%Prob.SolverInf = 'snopt';
%Prob.SolverInf = 'npsol';
%Prob.SolverInf = 'minos';
%Prob.SolverInf = 'conSolve';

% Set print level 2 to get output from PrintResult at the end
PriLev = 2;
Prob.PriLevOpt = 0;
Result  = tomRun('infSolve', Prob, PriLev);
```

19 MINIMAXLIN Problem

The **linear minimax** (**mimalin**) problem is defined as

$$\begin{array}{lrcccl} \min\limits_{x} & \max Dx & & & & \\ \text{subject to} & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \end{array} \tag{21}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $b_L, b_U \in \mathbb{R}^{m_1}$, $A \in \mathbb{R}^{m_1 \times n}$ and $D \in \mathbb{R}^{m_2 \times n}$.

The following file illustrates how to solve a linear minimax problem in TOMLAB.

File: tomlab/quickguide/minimaxlinQG.m

Open the file for viewing, and execute minimaxlinQG in Matlab.

```
% minimaxlinQG is a small example problem for defining and solving
% linear minimax programming problems using the TOMLAB format.

Name = 'Linear Minimax Test 1';
x_0  = [1;1;1;1];          % Initial value
x_L  = [-10;-10;-10;-10];  % Lower bounds on x
x_U  = [10;10;10;10];      % Upper bounds on x

% Solve the problem min max Dx while eliminating abs for the final two
% residuals by adding them with reverse signs.
% i.e. min max [D_1; D_2; D_3; -D_2; -D_3];
D = [9 8 7 6; -4 5 -6 -2; 3 4 5 -6; 4 -5 6 2; -3 -4 -5 6]; % D Matrix

% Add the linear constraint -x(1) + x(2) + 2 >= 0
% Write the constraint as x(1) - x(2) <= 2

% The A matrix could be specified dense or sparse
% A   = sparse([1 -1 0 0]);

A   = [1 -1 0 0];
b_L = -inf;
b_U = 2;

c = zeros(4,1); % Dummy objective

% Generate an LP problem using the Tomlab Quick format
% Use mipAssign if solving a mixed-integer problem
Prob = lpAssign(c, A, b_L, b_U, x_L, x_U, x_0, Name);
Prob.QP.D = D;

Prob.f_Low = 0;
Prob.SolverInf = 'minos';

% One may set other solvers:
```

```
% Prob.SolverInf = 'cplex';
% Prob.SolverInf = 'xa';
% Prob.SolverInf = 'snopt';
% Prob.SolverInf = 'milpSolve';

% Set print level 1 to get output from PrintResult at the end
PriLev = 1;
Prob.PriLevOpt = 0;

Result  = tomRun('infLinSolve', Prob, PriLev);
```

20 L1 Problem

The **constrained L1** (**L1**) problem is defined as

$$\begin{array}{lllll} \min\limits_{x} & \sum_i |r_i(x)| & & & \\ \text{subject to} & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{22}$$

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

The L1 solution can be obtained by the use of any suitable nonlinear TOMLAB solver.

The following files define a problem in TOMLAB.

File: tomlab/quickguide/L1QG_r.m, L1QG_J.m

```
r:   Residual vector
J:   Jacobian matrix
```

The following file illustrates how to solve an L1 problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/L1QG.m

Open the file for viewing, and execute L1QG in Matlab.

```
% L1QG is a small example problem for defining and solving
% an L1 problem using the TOMLAB format.

Name = 'Madsen-Tinglett 1';
x_0  = [1;1];        % Initial value
x_L  = [-10;-10];    % Lower bounds on x
x_U  = [10;10];      % Upper bounds on x

% Solve the problem min max |r_i(x)|, where i = 1,2,3
% Solve the problem by eliminating abs, doubling the residuals, reverse sign
% i.e. min max [r_1; r_2; r_3; -r_1; -r_2; -r_3];
y = [-1.5; -2.25; -2.625]; % The data values
t = [];                 % No time vector used

% Generate the problem structure using the Tomlab format
% as a standard least squares problem.
%
% The part in the residuals dependent on x are defined in L1ex_r.m
% The Jacobian is defined in L1ex_J.m

Prob = clsAssign('L1QG_r', 'L1QG_J', [], x_L, x_U, Name, x_0, y, t);

% Set JacPattern, L1Solve will compute correct ConsPattern
Prob.JacPattern  = ones(length(y),2);
```

```
% Set the optimal values into the structure (for nice result presenting)

Prob.x_opt=[3, 0.5];
Prob.f_opt=0;

% Use the standard method in L1Solve
Prob.L1Type = 1;

% Get the default solver
Solver = GetSolver('con',1,0);

Prob.SolverL1 = Solver;

% One may set the solver directly:
%Prob.SolverL1 = 'snopt';
%Prob.SolverL1 = 'npsol';
%Prob.SolverL1 = 'minos';
%Prob.SolverL1 = 'conSolve';

% These statements generate ASCII log files when running SNOPT
if strcmpi(Solver,'snopt')
   Prob.SOL.PrintFile = 'snopt.txt';
   Prob.SOL.SummFile  = 'snopts.txt';
   Prob.SOL.optPar(1) = 10;   % Print level in SNOPT
   %Prob.SOL.optPar(13) = 3;   % Verify level in SNOPT
end

% Set print level 2 to get output from PrintResult at the end
PriLev = 2;
Prob.PriLevOpt = 0;

Result  = tomRun('L1Solve', Prob, PriLev);
```

21 L1LIN Problem

The linearly **constrained L1LIN** (**L1LIN**) problem is defined as

$$\begin{array}{rl} \min\limits_{x} & |Cx - y)| + alpha * |Lx| \\ \text{subject to} & x_L \leq x \leq x_U \\ & b_L \leq Ax \leq b_U \end{array} \tag{23}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $b_L, b_U \in \mathbb{R}^{m_2}$, $A \in \mathbb{R}^{m_1 \times n}$, $C \in \mathbb{R}^{m_2 \times n}$, $y \in \mathbb{R}^{m_2}$, $L \in \mathbb{R}^{n \times b}$ and $alpha \in \mathbb{R}^1$.

The L1Lin solution can be obtained by the use of any suitable linear TOMLAB solver.

The following file illustrates how to solve an L1Lin problem in TOMLAB.

File: tomlab/quickguide/L1LinQG.m

Open the file for viewing, and execute L1LinQG in Matlab.

```
% L1LinQG is a small example problem for defining and solving
% a linearly constrained linear L1 problem using the TOMLAB format.

Name='L1LinSolve test example';      % Problem name, not required.
n = 6;
x_L = -10*ones(n,1);                 % Lower bounds on x
x_U =  10*ones(n,1);                 % Upper bounds on x
x_0 = (x_L + x_U) / 2;               % Starting point

C = spdiags([1 2 3 4 5 6]', 0, n, n); % C matrix
y = 1.5*ones(n,1);                    % Data vector

% Matrix defining linear constraints
A = [1 1 0 0 0 0];
b_L = 1;                  % Lower bounds on the linear inequalities
b_U = 1;                  % Upper bounds on the linear inequalities

% Defining damping matrix
Prob.LS.damp = 1;
Prob.LS.L = spdiags(ones(6,1)*0.01, 0, 6, 6);

% See 'help llsAssign' for more information.
Prob = llsAssign(C, y, x_L, x_U, Name, x_0, ...
                           [], [], [], ...
                           A, b_L, b_U);

Prob.SolverL1 = 'lpSimplex';
Result = tomRun('L1LinSolve', Prob, 1);
% Prob.SolverL1 = 'MINOS';
% Result = tomRun('L1LinSolve', Prob, 1);
% Prob.SolverL1 = 'CPLEX';
% Result = tomRun('L1LinSolve', Prob, 1);
```

22 LINRAT Problem

The linearly **constrained linear ratio** (**LINRAT**) problem is defined as

$$\begin{array}{lrcccl} \min\limits_{x} & \frac{c1x}{c2x} & & & & \\ \text{subject to} & x_L & \leq & x & \leq & x_U \\ & b_L & \leq & Ax & \leq & b_U \end{array} \tag{24}$$

where $c1, c2, x, x_L, x_U \in \mathbb{R}^n$, $b_L, b_U \in \mathbb{R}^{m_2}$, and $A \in \mathbb{R}^{m_1 \times n}$.

The LINRAT solution can be obtained by the use of any suitable linear TOMLAB solver.

The following file illustrates how to solve a LINRAT problem in TOMLAB.

File: tomlab/quickguide/linratQG.m

Open the file for viewing, and execute linratQG in Matlab.

```
% linratQG is a small example problem for defining and solving
% linear ratio programming problems using the TOMLAB format.

Name = 'Linear Ratio 1';
x_0  = [2;2;2;2];          % Initial value
x_L  = [1;2;3;4];          % Lower bounds on x
x_U  = [100;100;50;50];    % Upper bounds on x

% Define the numerator and denominator for the objective
c1 = [3;7;9;11];
c2 = [20;15;10;5];

% Add the linear constraint x(1) + x(2) + x(3) + x(4) - 20 <= 0
% Write the constraint as x(1) + x(2) + x(3) + x(4) <= 20

% The A matrix could be specified dense or sparse
% A   = sparse([1 1 1 1]);

A   = [1 1 1 1];
b_L = -inf;
b_U = 20;

c = zeros(4,1); % Dummy objective

% Generate an LP problem using the Tomlab Quick format
% Use mipAssign if solving a mixed-integer problem
Prob = lpAssign(c, A, b_L, b_U, x_L, x_U, x_0, Name);
Prob.QP.c1 = c1;
Prob.QP.c2 = c2;

Prob.SolverRat = 'minos';
```

```
% One may set other solvers:
% Prob.SolverRat = 'cplex';
% Prob.SolverRat = 'xa';
% Prob.SolverRat = 'snopt';
% Prob.SolverRat = 'milpSolve';

% Set print level 1 to get output from PrintResult at the end
PriLev = 1;
Prob.PriLevOpt = 0;

Result  = tomRun('linRatSolve', Prob, PriLev);
```

23 GOAL Problem

The **constrained goal attainment** (**goal**) problem is defined as

$$\begin{array}{lllll} \min\limits_{x} & \max & lam : r(x) - w * lam \leq g & & \\ \text{subject to} & x_L & \leq \quad x \quad \leq & & x_U \\ & b_L & \leq \quad Ax \quad \leq & & b_U \\ & c_L & \leq \quad c(x) \quad \leq & & c_U \end{array} \tag{25}$$

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

The goal solution can be obtained by the use of any suitable nonlinear TOMLAB solver.

The following files define a problem in TOMLAB.

File: tomlab/quickguide/goalsQG_r.m, goalsQG_J.m, goalsQG_c, goalsQG_dc

```
r:   Residual vector
J:   Jacobian matrix
c:   Nonlinear constraint vector
dc:  Nonlinear constraint gradient matrix
```

The following file illustrates how to solve a goal attainment problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/goalsQG.m

Open the file for viewing, and execute goalsQG in Matlab.

```
% goalsQG is a small example problem for defining and solving
% multi criteria optimization problems using the TOMLAB format.

Name='EASY-TP355';
% Constrained least squares problem, four quadratic terms and local solutions
% Hock W., Schittkowski K. (1981):
x_0 = zeros(4,1);    % Lower bounds for x.
x_L = zeros(4,1);    % Upper bounds for x.
x_U = 1e5*ones(4,1); % Starting point.
x_min = [];          % For plotting.
x_max = [];          % For plotting.
A   = [1 0 0 0;0 1 0 0];  % Linear constraints.
b_L = [0.1;0.1];          % Lower bounds.
b_U = [0.1;0.1];          % Upper bounds.
c_L = 0;                  % Lower bounds.
c_U = 0;                  % Upper bounds.
y   = zeros(2,1);         % Residuals

Prob = clsAssign('goalsQG_r', 'goalsQG_J', [], x_L, x_U, Name, x_0,...
                 y, [], [], [], [], [],...
                 A, b_L, b_U, 'goalsQG_c', 'goalsQG_dc', [], c_L, c_U,...
                 x_min, x_max);
```

```
PriLev = 2;
Result = tomRun('goalSolve', Prob, PriLev);
```

24 SIM Problem

Simulation problems can be of any problem type, but in general they are global black-box problem interfacing an external simulator. The setup may require that objective values and constraints are evaluated simultaneously. To accommodate this in TOMLAB a special assign routine has been developed, *simAssign*. The solver execution will be much more efficient if using this assign routine.

The simulation problem is identical to the mixed-integer nonlinear programming problem defined as:

$$\begin{array}{lccccc} \min\limits_{x} & f(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{26}$$

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

Example problem:

The following files define a problem in TOMLAB.

File: tomlab/quickguide/simQG.m, simQG_fc.m, simQG_gdc.m

```
fc:   Function value and nonlinear constraint vector
gdc:  Gradient vector and nonlinear constraint gradient matrix
```

The following file illustrates how to solve this simulation (SIM) problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/simQG.m

Open the file for viewing, and execute simQG in Matlab.

```
% simQG is a small example problem for defining and solving simulation
% problems where the objective function and constraints are evaluated
% during the same function call.

Name = 'HS 47';
b_L  = [];
b_U  = [];
A    = [];
c_L  = [0; 0; 0];
c_U  = [0; 0; 0];
x_0  = [2; sqrt(2); -1; 2-sqrt(2); .5];
x_L  = [];
x_U  = [];

Prob = simAssign('simQG_fc', 'simQG_gdc', [], [], x_L, x_U, ...
       Name, x_0, [], A, b_L, b_U, [], c_L, c_U);
```

```
Result = tomRun('snopt', Prob, 1);
% Prob.KNITRO.options.HESSOPT = 6;
% Prob.KNITRO.options.ALG = 3;
% Result2 = tomRun('knitro', Prob, 1);
```

25 GP Problem

Geometric programming problems are a set of special problems normally solved with TOMLAB /GP. The optimum is commonly non-differentiable. Problems are modeled on the primal form, but the dual is entered and solved.

The primal geometric programming problem is defined as:

$$(GP) \quad V_{GP} := \begin{array}{ll} \text{minimize} & g_0(t) \\ \text{subject to} & g_k(t) \leq 1, \quad k = 1, 2, \ldots, p \\ & t_i > 0, \qquad i = 1, 2, \ldots, m \end{array} \tag{27}$$

where

$$g_0(t) = \sum_{j=1}^{n_0} c_j t_1^{a_{1j}} ... t_m^{a_{mj}} \tag{28}$$

$$g_k(t) = \sum_{j=n_{k-1}+1}^{n_k} c_j t_1^{a_{1j}} ... t_m^{a_{mj}}, \quad k = 1, 2, \ldots, p. \tag{29}$$

Given exponents a_{ij} for the ith variable in the jth product term, $i = 1, \ldots, m$ and $j = 1, \ldots, n_p$, are arbitrary real constants and term coefficients c_j are positive.

Example problem:

$$(P1) \quad \begin{array}{ll} \min & 5x_1 + 50000x_1^{-1} + 20x_2 + 72000x_2^{-1} + 10x_3 + 144000x_3^{-1} \\ \text{subject to} & 4x_1^{-1} + 32x_2^{-1} + 120x_3^{-1} <= 1 \\ & x \geq 0 \end{array}$$

The following file defines and solves the problem in TOMLAB.

File: tomlab/quickguide/gpQG.m

Open the file for viewing, and execute gpQG in Matlab.

```
% gpQG is a small example problem for defining and solving
% geometric programming problems using the TOMLAB format.

nterm = [6;3];
coef = [.5e1;.5e5;.2e2;.72e5;.1e2;.144e6;.4e1;.32e2;.12e3];
A = sparse([ 1  -1  0   0  0  0 -1  0  0;...
             0   0  1  -1  0  0  0 -1  0;...
             0   0  0   0  1 -1  0  0 -1])';

Name  = 'GP Example';  % File gpQG.m

% Assign routine for defining a GP problem.
Prob = gpAssign(nterm, coef, A, Name);
```

```
% Calling driver routine tomRun to run the solver.
% The 1 sets the print level after optimization.

Result = tomRun('GP', Prob, 1);
```

26 LCP Problem

The general formulation in TOMLAB for a linear complementarity problem is:

$$\begin{array}{lll} \min\limits_{x} & f(x) = c^T x & \\ & & \\ & x_L \leq x \leq x_U & \\ s/t & b_L \leq Ax \leq b_U & \end{array} \tag{30}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. Equality constraints are defined by setting the lower bound to the upper bound.

The complementarity conditions can be any combination of decision variables and linear constraints:

- $x(i) \perp x(j)$
- $x(i) \perp A(j,:)x$
- $A(i,:)x \perp A(j,:)x$

Example problem:

The following file defines a problem in TOMLAB.

File: tomlab/quickguide/lcpQG.m

Open the file for viewing, and execute lcpQG in Matlab.

```
% lcpQG is a small linear complementary quick guide example
%
% minimize f: 2*x(1) + 2*x(2) - 3*x(3) - 3*x(4) - 60;
%
% Variable bounds:
%    x(1:2) >= 0, <= 50;
%    x(3:4) unbounded
%    x(5:10) >= 0;
%
% subject to:
%
%    c1: x(1) + x(2) + x(3) - 2*x(4) - 40 <= 0;
%
%    F1: 0 = 2*x(3) - 2*x(1) + 40 - (x(5) - x(6) - 2*x(9));
%    F2: 0 = 2*x(4) - 2*x(2) + 40 - (x(7) - x(8) - 2*x(10));
%
%    g1: 0 <= x(3) + 10           complements  x(5) >= 0;
%    g2: 0 <= -x(3) + 20          complements  x(6) >= 0;
%    g3: 0 <= x(4) + 10           complements  x(7) >= 0;
```

```
%    g4: 0 <= -x(4) + 20           complements  x(8) >= 0;
%    g5: 0 <= x(1) - 2*x(3) - 10   complements  x(9) >= 0;
%    g6: 0 <= x(2) - 2*x(4) - 10   complements  x(10) >= 0;
%
% These constraints above are written as 0 <= c(x) but we have to
% move the constant terms out of the linear expressions:
%
%    c1: x(1) + x(2) + x(3) - 2*x(4) <= 40
%
%    F1: -40 =  2*x(3) - 2*x(1) - x(5) + x(6) + 2*x(9);
%    F2: -40 =  2*x(4) - 2*x(2) - x(7) + x(8) + 2*x(10);
%
%    g1: -10 <= x(3)             complements  x(5) >= 0;
%    g2: -20 <= -x(3)            complements  x(6) >= 0;
%    g3: -10 <= x(4)             complements  x(7) >= 0;
%    g4: -20 <= -x(4)            complements  x(8) >= 0;
%    g5:  10 <= x(1) - 2*x(3)    complements  x(9) >= 0;
%    g6:  10 <= x(2) - 2*x(4)    complements  x(10) >= 0;
%
% An MPEC from F. Facchinei, H. Jiang and L. Qi, A smoothing method for
% mathematical programs with equilibrium constraints, Universita di Roma
% Technical report, 03.96. Problem number 7

Name = 'bilevel1';

% Number of variables:   10
% Number of constraints: 9

x_L = zeros(10,1);
x_L(3:4) = -inf;

x_U = inf*ones(size(x_L));
x_U(1:2) = 50;

c = [2 2 -3 -3 0 0 0 0 0 0 ]';

mpec = sparse( [
    5     0     4     0     0     0
    6     0     5     0     0     0
    7     0     6     0     0     0
    8     0     7     0     0     0
    9     0     8     0     0     0
    10    0     9     0     0     0]);

b_L = [-inf, -40, -40, -10, -20, -10, -20, 10, 10 ]';
b_U = [ 40, -40, -40, inf, inf, inf, inf, inf, inf]';
```

```
A =[
    1     1     1    -2     0     0     0     0     0     0
   -2     0     2     0    -1     1     0     0     2     0
    0    -2     0     2     0     0    -1     1     0     2
    0     0     1     0     0     0     0     0     0     0
    0     0    -1     0     0     0     0     0     0     0
    0     0     0     1     0     0     0     0     0     0
    0     0     0    -1     0     0     0     0     0     0
    1     0    -2     0     0     0     0     0     0     0
    0     1     0    -2     0     0     0     0     0     0
    ];

x_0 = ones(10,1);

Prob = lcpAssign(c, x_L, x_U, x_0, A, b_L, b_U, mpec, Name);

Prob.KNITRO.options.ALG = 3;
Prob.PriLevOpt = 2;
Result = tomRun('knitro',Prob,2);

x = Result.x_k;

% The slack values:
s = x(11:end)
x = x(1:10)

A = Prob.orgProb.A;
Ax = A*x;

% The last 6 elements of Ax, on the original form is:

Ax1 = Ax(4:9) + [10,20,10,10,-10,-10]'

% ... in a scalar product with the corresponding elements of x,
% should be zero:

l = x(5:10)
Ax1'*l
```

27 QCP Problem

The general formulation in TOMLAB for a quadratic complementarity problem is:

$$\begin{array}{lll} \min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x & \\ s/t & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \end{array} & \end{array} \tag{31}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. Equality constraints are defined by setting the lower bound equal to the upper bound, i.e. for constraint i: $b_L(i) = b_U(i)$. Fixed variables are handled the same way.

The complementarity conditions can be any combination of decision variables and linear constraints:

- $x(i) \perp x(j)$
- $x(i) \perp A(j,:)x$
- $A(i,:)x \perp A(j,:)x$

The following file defines a problem in TOMLAB.

File: tomlab/quickguide/qcpQG.m

Open the file for viewing, and execute qcpQG in Matlab.

```
% qcpQG is a small quadratic complementary quick guide example
%
% minimize  f = (x(1)-5)^2 + (2*x(2)+1)^2
%
% subject to
%
%    2*(x(2)-1) - 1.5*x(1) + x(3) - 0.5*x(4) + x(5) == 0   % c(1)
%     3*x(1) - x(2) - 3     >= 0                           % c(2)
%      -x(1) + 0.5*x(2) + 4 >= 0                           % c(3)
%      -x(1) -     x(2) + 7 >= 0                           % c(4)
%
%    x(1:5) >= 0
%
%    Complementarity conditions:
%
%    x(3) _|_ c(2)
%    x(4) _|_ c(3)
%    x(5) _|_ c(4)
```

```
%  If we omit the constant term 27 from f(x), we can write f = x'*F*x + c'*x:

F      = zeros(5,5);
F(1,1) = 1;
F(2,2) = 4;
c      = [-10,4,0,0,0]';

% Linear constraints, by moving constant terms in c1-c4 to the right hand
% side:
A = [...
    -1.5   2.0  1.0  -0.5  1.0 ; ...
     3.0  -1.0  0     0    0   ; ...
    -1.0   0.5  0     0    0   ; ...
    -1.0  -1.0  0     0    0 ];

b_L = [ 2 , 3   , -4  , -7 ]';
b_U = [ 2 , inf , inf , inf]';

% Lower and upper bounds
x_L =    zeros(5,1);
x_U = inf*ones(5,1);
x_0 = [];

Name = 'JF-BARD-1998-QP';

% Complementarity pairs:
mpec = [ ...
        3,0, 2,0, 0,0 ; ...
        4,0, 3,0, 0,0 ; ...
        5,0, 4,0, 0,0 ]

% Assign a TOMLAB 'qp' problem:
Prob = qcpAssign(F, c, A, b_L, b_U, x_L, x_U, x_0, mpec, Name);

% Three slacks have been added to the problem, easy to see by looking at the
% new linear constraint matrix:
A = Prob.A

% and the original linear constraint matrix:
A_orig = Prob.orgProb.A

% Enable some crossover iterations, to "polish" the solution a bit in
% case KNITRO chooses an interior point algorithm:

Prob.KNITRO.options.MAXCROSSIT = 100;
Prob.PriLevOpt = 2;
```

```
% Solve the QP (with MPEC pairs) using KNITRO:
Result = tomRun('knitro',Prob,2);

x = Result.x_k

% Values of slacks that were added by BuildMPEC
s = x(6:8)

% Original A * original variables, subtract the constants in the
% constraints to get the result on the c(x) >= 0 form

ax = A(:,1:5) * x(1:5) - b_L

% These are now complementary:
ax(2:4), x(3:5)

% Should be zero, or very close to zero:
ax(2:4)'*x(3:5)
```

28 MCP Problem

TOMLAB requires that general nonlinear complementarity problems are defined in Matlab m-files. The function to be optimized does not need to be supplied (pure equilibrium problem). It is recommended that the user supply as many analytical functions as possible.

The constrained nonlinear complementarity problem is defined as:

$$\begin{array}{lrcccl} \min\limits_{x} & f(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{32}$$

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

The complementarity conditions can be any combination of decision variables and linear/nonlinear constraints:

- $x(i) \perp x(j)$
- $x(i) \perp A(j,:)x$
- $x(i) \perp c(j)$
- $A(i,:)x \perp A(j,:)x$
- $A(i,:)x \perp c(j)$
- $c(i) \perp c(j)$

The following files define the problem in TOMLAB.

File: tomlab/quickguide/mcpQG_f.m, mcpQG_g.m, mcpQG_H.m, mcpQG_c.m, mcpQG_dc.m, mcpQG_d2c.m

```
f:   Function value
g:   Gradient vector
H:   Hessian matrix
c:   Nonlinear constraint vector
dc:  Nonlinear constraint gradient matrix
d2c: The second part of the Hessian to the Lagrangian function for the nonlinear constraints.
```

The following file illustrates how to solve a MCP problem in TOMLAB. Also view the m-files specified above for more information.

File: tomlab/quickguide/mcpQG.m

Open the file for viewing, and execute mcpQG in Matlab.

```
% mcpQG.m
%
% Demonstrates how to setup and solve a nonlinear problem with equilibrium
% (complementary) constraints.
%
% TOMLAB /KNITRO is capable of solving general complementarity problems
%
% The problem is the following:
%
% minimize  f = (x(1)-5)^2 + (2*x(2)+1)^2
%
% subject to
%
%    2*(x(2)-1) - 1.5*x(1) + x(3) - 0.5*x(4) + x(5) == 0   % c(1)
%     3*x(1) - x(2) - 3     >= 0                           % c(2)
%      -x(1) + 0.5*x(2) + 4 >= 0                           % c(3)
%      -x(1) -     x(2) + 7 >= 0                           % c(4)
%
%    x(1:5) >= 0
%
% The following complementarity demands are also imposed:
%
%   0 <= c(2) _|_ x(3) >= 0
%   0 <= c(3) _|_ x(4) >= 0
%   0 <= c(4) _|_ x(5) >= 0
%
% All four constraints are in fact linear, but are modelled as nonlinear
% constraints with the constants moved to the LHS.
%
% The same problem is solved as a explicitly quadratic complementarity
% problem in qcpQG.m

% There are no (explicitly) linear constraints.
A   = [];
b_L = [];
b_U = [];

% Provide a pattern for the nonlinear constraints
ConsPattern = [
    1 1 1 1 1
    1 1 1 0 0
    1 1 0 0 0
    1 1 0 0 0
    ];

HessPattern = [];
```

```
% First constraint is equality == 0, the remaining are >= 0
c_L = [0,0,0,0]';
c_U = [0,inf,inf,inf]';

x_L = zeros(5,1);
x_U = inf*ones(5,1);
x_0 = x_L;

Name = 'JF Bard 1998 MPEC';

% Functions for calculating the nonlinear function and derivative values.
f   = 'mcpQG_f';
g   = 'mcpQG_g';
H   = 'mcpQG_H';
c   = 'mcpQG_c';
dc  = 'mcpQG_dc';
d2c = 'mcpQG_d2c';

% Each row of mpec is one pair. The first says x(3) _|_ c(2) should be
% complementary. Exactly two nonzeros per row is allowed.

mpec = [ ...
    3,0, 0,0, 2,0; ...
    4,0, 0,0, 3,0; ...
    5,0, 0,0, 4,0; ...
    ];

Prob = mcpAssign(f, g, H, HessPattern, x_L, x_U, Name, x_0, mpec, ...
    [], ...
    A, b_L, b_U, c, dc, d2c, ConsPattern, c_L, c_U);

Prob.PriLevOpt = 2;

% KNITRO options. Algorithm 3 works good on this problem.
opts     = [];
opts.ALG = 3;
Prob.KNITRO.options = opts;

% Now solve this problem with KNITRO:
Result = tomRun('knitro',Prob,2);

% The original problem is available as Prob.orgProb.

N0 = Prob.orgProb.N;

x = Result.x_k;
s = x(N0+1:end)     % The slacks added by BuildMPEC
```

```
x = x(1:N0)         % The "original" variables

% Constraint values for the modified problem. If not infeasible, these
% should all be zero because BuildMPEC has changed the constraints in the
% mpec pairs to include slack variables.

c = Result.c_k

% The permutation matrix for the slacks in the nonlinear constraints is
% available:

MP = full(Prob.MPEC.MP);

% The values of the constraints + their respective slacks:

C = c + MP*s

x(3:5), C(2:4)

% This should give something like:
%
% ans =
%
%     3.5000
%          0
%          0
%
%
% ans =
%
%      0
%      3
%      6
%
% I.e, we have the complementarity conditions satisfied.
```

29 EXP Problem

Exponential fitting problems are a set of special problems normally solved by *expSolve*. Several different models are treated in TOMLAB:

Table 1: Exponential models treated in TOMLAB.

$f(t) = \sum_i^p \alpha_i e^{-\beta_i t}$,	$\alpha_i \geq 0$,	$0 \leq \beta_1 < \beta_2 < ... < \beta_p$.
$f(t) = \sum_i^p \alpha_i (1 - e^{-\beta_i t})$,	$\alpha_i \geq 0$,	$0 \leq \beta_1 < \beta_2 < ... < \beta_p$.
$f(t) = \sum_i^p t\alpha_i e^{-\beta_i t}$,	$\alpha_i \geq 0$,	$0 \leq \beta_1 < \beta_2 < ... < \beta_p$.
$f(t) = \sum_i^p (t\alpha_i - \gamma_i) e^{-\beta_i t}$,	$\alpha_i, \gamma_i \geq 0$,	$0 \leq \beta_1 < \beta_2 < ... < \beta_p$.
$f(t) = \sum_i^p t\alpha_i e^{-\beta_i (t-\gamma_i)}$,	$\alpha_i \geq 0$,	$0 \leq \beta_1 < \beta_2 < ... < \beta_p$.

Example problem:

The vectors t, y contain the following data:

t_i	0	1.00	2.00	4.00	6.00	8.00	10.00	15.00	20.00
y_i	905.10	620.36	270.17	154.68	106.74	80.92	69.98	62.50	56.29

Setup and solve the problem of fitting the data to a two-term exponential model:

$$f(t) = \alpha_1 e^{-\beta_1 t} + \alpha_2 e^{-\beta_2 t},$$

The following file defines and solves the problem in TOMLAB.

File: tomlab/quickguide/expQG.m

Open the file for viewing, and execute expQG in Matlab.

```
% expQG is a small example problem for defining and solving exponential
% fitting problems using the TOMLAB format. See the TOMLAB manual for
% options.

% f(t) = alpha(1)*exp(-beta(1)*t) + alpha(2)*exp(beta(2)*t)

t = [0 1.00 2.00 4.00 6.00 8.00 10.00 15.00 20.00]';
y = [905.10 620.36 270.17 154.68 106.74 80.92 69.98 62.50 56.29]';
p      = 2;                          % Two terms
Name   = 'Simple two-term exp fit'; % Problem name, can be anything
wType  = 0;                          % No weighting
SepAlg = 0;                          % Separable problem
Prob = expAssign(p,Name,t,y,wType,[],SepAlg);
```

```
Prob.SolverL2 = 'nlssol';
Result = tomRun('expSolve',Prob,1);
%Prob.SolverL2 = 'snopt';
%Result = tomRun('expSolve',Prob,1);
```

30 QPBLOCK Problem

The general formulation in TOMLAB for a quadratic programming problem is:

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2} x^T F x + c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{33}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. Equality constraints are defined by setting the lower bound equal to the upper bound, i.e. for constraint i: $b_L(i) = b_U(i)$. Fixed variables are handled the same way.

When using a general nonlinear solver such as SNOPT, KNITRO or CONOPT it is possible to define the objective on a block structure. The objective can take the following formats.

Case 1:

```
The quadratic objective can be separated into main sparse matrix F and
one or more additional two part blocks.

      min    0.5 * x' * F * x + d' * x +
             0.5 * x' * Fb.out' * Fb.inn * Fb.out * x (for i=1...p)
```

Case 2:

```
The quadratic objective can be separated into a main sparse matrix F and
two outer blocks.

      min    0.5 * x' * F * x + d' * x +
             0.5 * x' * Fb.out' * Fb.out * x (for i=1...p)
```

Case 3:

```
Case number 1 above, but F is supplied as a nx1 vector (diagonal)
```

Case 4:

```
Case number 2 above, but F is supplied as a nx1 vector (diagonal)
```

The following file defines a test case in TOMLAB.

File: tomlab/quickguide/qpblockQG.m

Open the file for viewing, and execute qpblockQG in Matlab.

```
% qpblockQG is a small example problem for defining and solving
% quadratic programming on a block structure using the TOMLAB format.
% See help qpblockAssign for more information.
Name  = 'QP Block Example';

switch 1
    case 1
        F     = [ 8   0
                  0   8 ];
        Fb(1).out = ones(3,2);
        Fb(1).inn = ones(3,3);
        Fb(2).out = ones(4,2);
        Fb(2).inn = ones(4,4);
        d     = [ 3  -4 ]';
    case 2
        F     = [ 8   0
                  0   8 ];
        Fb(1).out = ones(3,2);
        Fb(2).out = ones(4,2);
        d     = [ 3  -4 ]';
    case 3
        F     = [ 8   8]';
        Fb(1).out = ones(3,2);
        Fb(1).inn = ones(3,3);
        Fb(2).out = ones(4,2);
        Fb(2).inn = ones(4,4);
        d     = [ 3  -4 ]';
    case 4
        F     = [ 8   8]';
        Fb(1).out = ones(3,2);
        Fb(2).out = ones(4,2);
        d     = [ 3  -4 ]';
end

A     = [ 1   1        % Constraint matrix
          1  -1 ];
b_L   = [-inf  0  ]';  % Lower bounds on the linear constraints
b_U   = [  5   0  ]';  % Upper bounds on the linear constraints
x_L   = [  0   0  ]';  % Lower bounds on the variables
x_U   = [ inf inf ]';  % Upper bounds on the variables
x_0   = [  0   1  ]';  % Starting point
x_min = [-1 -1 ];      % Plot region lower bound parameters
x_max = [ 6  6 ];      % Plot region upper bound parameters
```

```
% Assign routine for defining a QP problem.
Prob = qpblockAssign(F, Fb, d, x_L, x_U, Name, x_0, A, b_L, b_U);

% Calling driver routine tomRun to run the solver.
% The 1 sets the print level after optimization.

Result = tomRun('snopt', Prob, 1);
% Result = tomRun('knitro', Prob, 1);
% Result = tomRun('conopt', Prob, 1);
```

31 Binary Selection Problems

The general formulation in TOMLAB for a binary selection problem (i.e. a problem that has binary variable products with other binary/interger/continuous variables) is:

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x^T F x + c^T x \\ s/t \quad & \begin{array}{rcccl} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U, \; x_j \in \mathbb{N} \quad \forall j \in \mathrm{I} \end{array} \end{aligned} \tag{34}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. The variables $x \in I$, the index subset of $1, ..., n$ are restricted to be integers. Equality constraints are defined by setting the lower bound equal to the upper bound, i.e. for constraint i: $b_L(i) = b_U(i)$. A subset of the variables x are binary. Another subset is the variable combinations with these binary variables.

The following files define two test cases in TOMLAB. The first case is a binary programming problem with binary combinations only. The second case has continuous variables (possible to replace by integers if needed) of which a subset are to be selected by enforcing a constraint on the binary variables.

File: tomlab/quickguide/binbinQG.m

Open the file for viewing, and execute binbinQG in Matlab.

```
% binbinQG is a small example problem for defining and solving
% combinatorial binary programming problems using the TOMLAB format.

Name  = 'binbinQG';

% The first 4 are for binary variables:
% b1 b2 b3 b4
% The rest are the combinations:
% b1*b2, b1*b3, b1*b4, b2*b3, b2*b4, b3*b4

% Coefficients in linear objective function
c     = [zeros(1,4) 1 2 3 2 4 3]';

% At least 3 binary variables are active
A     = [1 1 1 1 0 0 0 0 0 0];
b_U   = inf;
b_L   = 3;
x_L   = zeros(10,1);
x_U   = ones(10,1);

IntVars = ones(10,1);

Prob = mipAssign(c, A, b_L, b_U, x_L, x_U, [], Name, [], [], IntVars);

% Give the indices for the combinations and variables combined
Prob = binbin2lin(Prob, (5:10)', [1;1;1;2;2;3], [2;3;4;3;4;4]);
```

```
Result = tomRun('cplex', Prob, 1);
```

File: tomlab/quickguide/bincontQG.m

Open the file for viewing, and execute bincontQG in Matlab.

```
% bincontQG is a small example problem for defining and solving
% where a subset of the continuous variables are active.

Name  = 'bincontQG';

% The first n are for binary variables:
% b1 ... bn
%
% Variables n+1 to 2*n are continuous:
% c1 ... cn
%
% Variables 2*n+1 to 3*n are combinations:
% b1*c1 ... bn*cn

% Coefficients in linear objective function
n   = 20;
cov1 = magic(20)/100;
cov1 = cov1(:,1);
F   = [zeros(2*n,3*n); zeros(n,2*n), diag(cov1)];
c   = zeros(3*n,1);

% 10 variables to be selected
% Combined combinations greater than 1000
A     = [ones(1,n), zeros(1,2*n);
    zeros(1,2*n), ones(1,n)];
b_U   = [10;inf];
b_L   = [10;1000];
x_L   = zeros(3*n,1);
x_U   = [ones(n,1); 1e4*ones(2*n,1)];

IntVars = [ones(n,1); zeros(2*n,1)];

Prob = miqpAssign(F, c, A, b_L, b_U, x_L, x_U, [], IntVars);

% Give the indices for the combinations and variables combined
Prob = bincont2lin(Prob, (2*n+1:3*n)', (1:n)', (n+1:2*n)');

Result = tomRun('cplex', Prob, 1);
```

32 PIECE-WISE LINEAR Problem

The general formulation in TOMLAB for a piece-wise linear programming problem is:

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x^T F x + c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U,\ x_j \in \mathbb{N} \quad \forall j \in \mathrm{I} \end{array} \end{aligned} \tag{35}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. The variables $x \in I$, the index subset of $1, ..., n$ are restricted to be integers. Equality constraints are defined by setting the lower bound equal to the upper bound, i.e. for constraint i: $b_L(i) = b_U(i)$. A subset of the variables x are piece-wise linear.

Solving piece-wise linear problem is mainly recommended by using TOMLAB /CPLEX or similar solver.

The following file defines a test case in TOMLAB. It is possible to use two syntax variations when defining the problem (see help addPwLinFunc for more information).

File: tomlab/quickguide/piecewiseQG.m

Open the file for viewing, and execute piecewiseQG in Matlab.

```
% pieceWiseQG is a small example problem for defining and solving piece-
% wise linear programming problems using TOMLAB. The exmaple was taken from
% the ILOG CPLEX 11.0 User's Manual: a Transport Example.
%
% Objects must be shipped from supply locations to demand locations.
%
% Supply amounts (i): [1000 850 1250]
% Demand amounts (j): [900 1200 600 400]
%
% The model forces all demands to be satisfied and all supplies to be
% shipped.
%
% sum{i=1:nSupply} x(i,j) = supply(i)
% sum{j=1:nDemand} x(i,j) = demand(j)
%
% The cost (to be minimized) is defined as follows:
%
% sum{i=1:nSupply} sum{j=1:nDemand} cost(i,j)*x(i,j)
%
% The cost is a piece-wise linear function decribed by:
%
% Case 1 (non-convex):
% 120 per item, between 0 and 200
% 80 per item, between 200 and 400
% 50 per item, between 400 and above
%
% Case 2 (convex):
```

```
% 30 per item, between 0 and 200
% 80 per item, between 200 and 400
% 130 per item, between 400 and above

nDemand = 4;   % Demand nodes
nSupply = 3;   % Supply nodes
supply = [1000;850;1250];
demand = [900;1200;600;400];

N = nDemand*nSupply;
%supply1 -> demand, supply2 -> demand, supply3 -> demand

IntVars = (1:nDemand*nSupply)';

% All supplies shipped to demand centers
A1 = zeros(nSupply,N);
for i=1:nSupply
    A1(i,(i-1)*nDemand+1:i*nDemand) = 1;
end
b_L1 = supply;
b_U1 = supply;

% All supplies received at demand centers
A2 = zeros(nDemand,N);
for i=1:nDemand
    A2(i,i:demandNodes:N) = 1;
end
b_L2 = demand;
b_U2 = demand;

% Merge constraints
A = [A1;A2];
b_L = [b_L1;b_L2];
b_U = [b_U1;b_U2];

% Bounds on variables based on supply
x_L = zeros(N,1);
x_U = repmat(demand,nSupply,1);

% Also limited by demand
for i=1:nSupply
    idx = (i-1)*nDemand+1:i*nDemand;
    x_U(idx) = min(supply(i),x_U(idx));
end

% We need N extra variables. One for each piece-wise function.
% The sum of these is the total cost.
```

```
c = [zeros(N,1);ones(N,1)];
A = [A, zeros(size(A,1),N)];
x_L = [x_L; zeros(N,1)];
x_U = [x_U; inf*ones(N,1)];
x_0 = []; %No starting point possible
IntVars = [IntVars; zeros(N,1)];

Prob1 = mipAssign(c, A, b_L, b_U, x_L, x_U, x_0, 'PW-EX1', [], [], IntVars);
Prob2 = mipAssign(c, A, b_L, b_U, x_L, x_U, x_0, 'PW-EX2', [], [], IntVars);
Prob3 = mipAssign(c, A, b_L, b_U, x_L, x_U, x_0, 'PW-EX3', [], [], IntVars);
Prob4 = mipAssign(c, A, b_L, b_U, x_L, x_U, x_0, 'PW-EX4', [], [], IntVars);

% Syntax 1, non-convex
type = 'cplex';
point = [200;400];
slope = [120;80;50];
for i=1:N
    var    = i;   % Variable in piece-wise linear function
    funVar = N+i; % New variable to act as the new pw function.
    % We only need to specify the value of the function at one point,
    % f(a) = fa. This point and the slopes describe the function completely.
    % fa is zero when no units are shipped.
    a  = 0;
    fa = 0;
    Prob1 = addPwLinFun(Prob1, 1, type, var, funVar, point, slope, a, fa);
end

% Solve problem 1 (non-convex, syntax 1)
Result1 = tomRun('cplex', Prob1, 1);

% Syntax 2, non-convex
type = 'cplex';
firstSlope = 120;
point = [200;400];
value = [200*120;200*120+200*80];
lastSlope = 50;
for i=1:N
    var    = i;   % Variable in piece-wise linear function
    funVar = N+i; % New variable to act as the new pw function.
    Prob2 = addPwLinFun(Prob2, 2, type, var, funVar, firstSlope, point, ...
        value, lastSlope);
end

% Solve problem 1 (non-convex, syntax 2)
Result2 = tomRun('cplex', Prob2, 1);

% Syntax 1, convex
```

```
type = 'cplex';
point = [200;400];
slope = [30;80;130];
for i=1:N
    var    = i;
    funVar = N+i;
    a      = 0;
    fa     = 0;
    Prob3 = addPwLinFun(Prob3, 1, type, var, funVar, point, slope, a, fa);
end

% Solve problem 1 (convex, syntax 1)
Result3 = tomRun('cplex', Prob3, 1);

% Syntax 4, convex
type = 'cplex';
firstSlope = 30;
point = [200;400];
value = [200*30;200*30+200*80];
lastSlope = 130;
for i=1:N
    var    = i;
    funVar = N+i;
    a      = 0;
    fa     = 0;
    Prob4 = addPwLinFun(Prob4, 2, type, var, funVar, firstSlope, point, ...
        value, lastSlope);
end

% Solve problem 1 (convex, syntax 2)
Result4 = tomRun('cplex', Prob4, 1);

disp(sprintf('Syntax 1, non-convex results \n'));
disp(reshape(Result1.x_k(1:nDemand*nSupply,1),nDemand,nSupply));
disp(sprintf('Syntax 2, non-convex results \n'));
disp(reshape(Result2.x_k(1:nDemand*nSupply,1),nDemand,nSupply));
disp(sprintf('Syntax 1, convex results \n'));
disp(reshape(Result3.x_k(1:nDemand*nSupply,1),nDemand,nSupply));
disp(sprintf('Syntax 2, convex results \n'));
disp(reshape(Result4.x_k(1:nDemand*nSupply,1),nDemand,nSupply));
```

33 MAD Problem

TOMLAB /MAD is a package for general automatic differentiation of Matlab code. Usage is applicable for any applications needing derivatives. The package can be used standalone or as part of TOMLAB when floating point precision derivatives are needed.

Following is a simple example of standalone use:

```
>> x = 1;
>> x = fmad(x,1);
>> y = sin(x);
>> y

value =

    0.8415


derivatives =

    0.5403
```

An example problem with TOMLAB is included in the guide. The following file defines and solves two problems in TOMLAB.

File: tomlab/quickguide/madQG.m

Open the file for viewing, and execute madQG in Matlab.

```
% madQG are two examples for defining and solving nonlinear
% programming problems using TOMLAB /MAD

Name = 'RBB Problem';
x_0 = [-1.2 1]';     % Starting values for the optimization
x_L = [-10;-10];     % Lower bounds for x.
x_U = [2;2];         % Upper bounds for x.
fLowBnd = 0;         % Lower bound on function.

c_L = -1000;         % Lower bound on nonlinear constraints.
c_U = 0;             % Upper bound on nonlinear constraints.

Prob1 = conAssign('rbbQG_f', [], [], [], x_L, x_U, Name, x_0,...
                [], fLowBnd, [], [], [], 'rbbQG_c', [], [], [], c_L, c_U);

Prob2 = conAssign('rbbQG_f', 'rbbQG_g', [], [], x_L, x_U, Name, x_0,...
                [], fLowBnd, [], [], [], 'rbbQG_c', 'rbbQG_dc', [], [], c_L, c_U);
```

```
Prob1.Warning = 0;    % Turning off warnings.
Prob2.Warning = 0;    % Turning off warnings.

madinitglobals;
Prob1.ADObj  = 1; % Gradient calculated
Prob1.ADCons = 1; % Jacobian calculated
Result1 = tomRun('snopt', Prob1, 1);  % Only uses first order information.

madinitglobals;
Prob2.CONOPT.options.LS2PTJ = 0;
Prob2.ADObj  = -1; % Hessian calculated
Prob2.ADCons = -1; % Lagrangian function for the nonlinear constraints.
Result2 = tomRun('conopt', Prob2, 1);  % Uses second order information.
```

34 Important Information

Setting patterns is especially important for large-scale problems as memory needs to be managed more properly (a dense problem is normally assumed otherwise). Solver timings and recursive calls primarily applies to smaller problems but could also be important issues for larger test cases.

34.1 Passing addition variables

If the user wishes to pass additional variables to the user functions written the parameters need to be included in the *Prob* structure. The code snippets below illustrates how to do it.

In the main function where TOMLAB is called do:

```
Prob = *Assign(...);
Prob.user.a = a;
Prob.user.b = b;
Result = tomRun('solver', Prob, 1);
```

If for example the objective function needs the additional variables a and b do the following in the file:

```
function f = myobjective(x, Prob)
a = Prob.user.a;
b = Prob.user.b;
f = sum(x)*a + sum(x.^3)*b;
```

34.2 Using Patterns

For most problems it is critical to set the proper problem patterns (*ConsPattern*, *HessPattern* and *d2LPattern*) for memory allocation purposes and to speed up numerical differentiation. If analytical derivatives are given only the memory benefit will be seen.

See for example the minlpQG problem and use the following code:

```
minlpQG
Prob.ConsDiff = 1;
Result = tomRun('npsol', Prob, 1);
Prob.ConsDiff = 11;
Result = tomRun('npsol', Prob, 1);
```

As can be seen only 43 constraint evaluations are done for the second run with NPSOL. The reason being that the solver (numerical differentiation routines) can see which variables to perturb at the same time from the *ConsPattern*.

34.3 Solver Timings

Comparing different solvers with solution times under 5-10 seconds may yield incorrect results. When running a problem and a solver for the first time general overhead and loading of dll's consume the majority of the time. The following code illustrates the problem:

```
clear all
Prob = probInit('lp_prob', 1);
R = tomRun('minos', Prob, 1);
R = tomRun('minos', Prob, 1);
```

The first run may report a solution time around 0.3 seconds, while the second run shows that the real time spent on optimization is less than 0.01 seconds. When evaluating different solver solutions, all tests need to be run at least twice (more recommended).

There are ways to avoid the extra overhead associated with the driver routines *tomRun*. For example one can call the solver directly.

```
clear all
Prob = probInit('lp_prob', 1);
Prob = ProbCheck(Prob, 'minos');
R = minosTL(Prob);
PrintResult(R);
```

This is especially important when recursively calling the solver.

34.4 Recursive Calls

When doing recursive calls to a solver and modifying some of the inputs (not the size of the problem) one should call the solver as shown above. This minimizes the overhead during the solver call.

Warm start is commonly used when doing recursive calls. Several of the TOMLAB solvers support warm start.

```
clear all
Prob = probInit('lp_prob', 1);
R = tomRun('minos', Prob, 1);
Prob = WarmDefSOL('minos', Prob, R);
R = tomRun('minos', Prob, 1);
```

Similar code will work for MINOS (also LP-, QP-MINOS), SNOPT, SQOPT, NPSOL, NLSSOL, LPOPT, QPOPT, LSSOL.

When running the TOMLAB /MINLP solvers the following code is needed. Observe that only BQPD and filterSQP can be warm started, while miqpBB and minlpBB accept a starting point.

```
clear all
```

```
Prob = probInit('con_prob', 10);
R = tomRun('filterSQP', Prob, 1);
Prob = WarmDefDUNDEE('filterSQP', Prob, R);
R = tomRun('filterSQP', Prob, 1);
```

It is also possible to warm start TOMLAB /CPLEX for LP problems (only simplex solvers). When doing this one has to supply a basis. See 'help cplex' for more information.

The global solvers in the TOMLAB Base Module and TOMLAB /CGO are easily warm started. One simply sets *Prob.WarmStart = 1* before calling the solver the second round.

34.5 Verifying Problems

There are several routines and functionality in TOMLAB for verifying problem setup. *checkDerivs* can be used to check the absolute error of user supplied derivatives. *checkFuncs* make general user function checking and problem validation. One can also check the derivatives with for example SNOPT, MINOS and NPSOL. The code below illustrates how to generate a print file for further analysis.

```
clear all
Prob = probInit('con_prob', 10);
Prob.SOL.optPar(1)  = 111111;        % Major print level
Prob.SOL.optPar(2)  = 10;            % Minor print level
Prob.SOL.optPar(13) = 3;             % Verify level
Prob.SOL.PrintFile  = 'snoptP.txt';  % SNOPT print file name
Prob.SOL.SummFile   = 'snoptS.txt';  % SNOPT summary file name
R = tomRun('snopt', Prob, 1);
```

The print file *snoptP.txt* will provide details on the user supplied derivatives. Observe that this checking should never be done for production code. Once the derivatives have been verified it is recommended to set Prob.SOL.optPar(13) = -1 to avoid one extra function call.

34.6 Optimization Toolbox

There is an optimization toolbox interface included in `/tomlab/optim`. These routines can be used for quick testing of the TOMLAB capabilities when problems are setup for use with optimization toolbox. In general, the routines should be avoided as unnecessary overhead is introduced from the format conversion. If one has embedded calls they should be used as is.

34.7 Matlab functions

TOMLAB supports all types of Matlab functions; anonymous, nested and general sub-functions.

The following file illustrates various combinations of the options:

File: tomlab/quickguide/funchandleQG.m

Open the file for viewing, and execute funchandleQG; in Matlab.

```
% funchandleQG is a small example for defining and solving problems
% using nested functions, anonymous functions and subfunctions.

function [R1, R2, R3] = funchandleQG

Name='funchandleQG';

x_L = [0 0 0 0 0 ]';
x_U = [inf inf 1 1 1 ]';

A = [1 0     1  0 0 ; ...
    0 1.333 0  1 0 ; ...
    0 0    -1 -1 1 ];

b_L = [];
b_U = [1.6 ; 3 ; 0];

c_L = [1.25;3];
c_U = c_L;

x_0 = ones(5,1);

HessPattern = spalloc(5,5,0);
ConsPattern = [ 1 0 1 0 0; ...
    0 1 0 1 0 ];

% Example with local sub functions
Prob1 = minlpAssign(@my_f, @my_g, @my_H, HessPattern, ...
    x_L, x_U, Name, x_0, [], [], [], [], ...
    A, b_L, b_U, @my_c, @my_dc, @my_d2c, ...
    ConsPattern, c_L, c_U);

R1 = tomRun('knitro', Prob1, 1);

constr = @(x) [ x(1)^2+x(3) ; sqrt(x(2)^3)+1.5*x(4)];

% Example with local sub functions and anonymous function
% One directly into assign routine and one as variable input
Prob2 = minlpAssign(@(x) [2 3 1.5 2 -0.5]*x, @my_g, @my_H, HessPattern, ...
    x_L, x_U, Name, x_0, [], [], [], [], ...
    A, b_L, b_U, constr, @my_dc, @my_d2c, ...
    ConsPattern, c_L, c_U);

R2 = tomRun('knitro', Prob2, 1);
```

```
% Example with local sub functions, anonymous function
% and nested function
Prob3 = minlpAssign(@mynested_f, @my_g, @my_H, HessPattern, ...
    x_L, x_U, Name, x_0, [], [], [], [], ...
    A, b_L, b_U, constr, @my_dc, @my_d2c, ...
    ConsPattern, c_L, c_U);

    function f = mynested_f(x)
        f = [2 3 1.5 2 -0.5]*x;
    end

R3 = tomRun('knitro', Prob3, 1);

end

function f = my_f(x)
f = [2 3 1.5 2 -0.5]*x;
end

function g = my_g(x)
g = [2 3 1.5 2 -0.5]';
end

function H = my_H(x)
H = spalloc(5,5,0);
end

function c = my_c(x)
c = [ x(1)^2+x(3) ; sqrt(x(2)^3)+1.5*x(4)];
end

function dc = my_dc(x)
dc = [2*x(1)     0.0       1.0  0.0  0.0 ; ...
    0.0   1.5*sqrt(x(2)) 0.0  1.5  0.0];
end

function d2c = my_d2c(x,lam)
d2c      = spalloc(5,5,2);
d2c(1,1) = lam(1)*2;
d2c(2,2) = lam(2)*3/(4*sqrt(x(2)));
end
```