Practical Optimization with the TOMLAB Environment in Matlab

Professor Kenneth Holmström

Applied Optimization and Modeling Group (**TOM**)

Center for Mathematical Modeling
Department of Mathematics and Physics
Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden

September 15, 2001

Abstract

The TOMLAB /SOL v3.0 optimization environment is a powerful optimization tool in Matlab, which incooperates many results from the last 40 years of research in the field. More than 65 different algorithms for linear, discrete, global and nonlinear optimization are implemented in Matlab, and 14 Fortran solvers are integrated with the use of MEX file interfaces. It has been developed in cooperation with the SOL group at Stanford and UC San Diego and includes the SOL solvers SNOPT, NPSOL, MINOS, SQOPT, NLSSOL, QPOPT, LPOPT and LSQR. TOMLAB is available on Windows, Linux, SGI, HP, MAC, DEC and SUN systems.

This paper discusses the design and contents of TOMLAB, and some examples of its usage on practical real-life optimization problems. We emphasize the great importantance of using high-quality numerical software in the solution process of optimization problems. Global optimization methods are discussed, and the way they easily may be combined with local optimization methods to solve practical industrial and financial simulation and optimization problems.

More information on TOMLAB is available at the TOMLAB home page URL: **http://www.tomlab.net**. A 200 page User's Guide is available for download, together with demonstration versions of v3.0 /SOL, as well as v3.0 and v3.0 /MINI, which include a subset of the SOL solvers.

1 Introduction

The last decade has seen a tremendous growth in the use of MATLAB as an advanced tool for mathematical model building and analysis. The high-level interactive command language makes it easy to formulate any type of problem or algorithm, and basic linear algebra solutions for linear systems, eigenvalues and singular values are directly available. Numerical solutions to ordinary and partial differential equations are easily obtained. A number of language extensions in the form of toolboxes gives the user special solutions in very many fields, like finance, chemistry and control. The Simulink toolbox for mathematical simulation and the FEMLAB toolbox for multiphysics modeling are examples of highly advanced tools. As part of the advanced modeling, very often optimization problems of different kinds need to be solved. So far the support for optimization in MATLAB has been limited.

One major feature of traditional modeling languages is that they separate the model building from the solution process. Once the model is formulated, it is easy to solve the problem using any of the solvers provided for the particular type of optimization problem. The user should not have to bother about the actual solver interface and default values are provided which most often are sufficient to obtain the wanted solution.

Even if MATLAB is a very strong modeling language, what is missing is an uniform approach to the optimization solution process. The TOMLAB optimization environment attempts to fill this gap. It also improves the solver interaction, with the aid of a graphical user interface, and enables advanced use of callback algorithms in the solution of mixed-integer programs.

This paper presents some of the basic design ideas in TOMLAB and how a uniform format for optimization is obtained. Demonstration versions of TOMLAB are downloadable from the TOMLAB web site, URL: **http://www.tomlab.net**. Documentation, links to papers and information about solvers and features are also found at the site.

This paper is organized as follows. A brief overview of the development of TOMLAB is given in Section 2. In Section 3 the design of TOMLAB is discussed, with special emphasis on the modeling aspects of TOMLAB. The use of the graphical user interface (GUI) is shortly discussed. A few examples of the solution of constrained nonlinear problems illustrates the usage of TOMLAB. Section 4 presents examples on how to use TOMLAB to solve linear programming examples, and some further possibilities when solving linear and mixed-integer programming problems. Section 5 discusses the solution of nonlinear parameter estimation. The special treatment and applications of exponential sum fitting problems are presented in Section 6. A short discussion about global optimization is given in Section 7. More applications are presented in 8. Finally, some conclusions are given in Section 9.

2 The TOMLAB Development

The lack of good optimization tools in MATLAB motivated the early development of some of the solvers already in 1990. As the number of solvers and tools grew they were collected into two toolboxes, NLPLIB TB for nonlinear programming and parameter estimation [25], and OPERA TB for linear and discrete optimization [26]. The concept of TOMLAB was born in 1997.

The first TOMLAB version 1.0 was presented in Holmström [16, 15]. It handled small and medium size dense problems and was free for academic use. A User's Guide for TOMLAB v1.0 is available [27]. The system was quite large; only the Matlab code consisted of more than 300 M-files and more than 55,000 lines of code. Around one thousand persons downloaded the software during the year it was distributed.

The next version 2.0, presented in Holmström [20, 18], also handled large, sparse problems and the code was possible to automatically convert to C++ and compile and run faster than in MATLAB using the Mideva system. A User's Guide for TOMLAB v2.0 [22] describes the system in detail, now more than 80,000 lines of Matlab code in 420 files. Mideva is no longer distributed.

The TOMLAB system has been totally revised for the current version 3.0. Besides around 95,000 lines of code in about 450 files, now TOMLAB has around fifteen Fortran solvers interfaced. Portability issues have been solved and the full TOMLAB is now available on PC Windows, Linux and MAC, as well as Unix systems on Sun, DEC, HP and SGI machines.

TOMLAB is used for several different purposes. In the Applied Optimization Group at Mälardalen University, it has been used to develop algorithms and solve a number of applied optimization projects [17, 4, 12]. At the Stanford System Optimization Laboratory it is used as a subproblem solver in the development of new numerical algorithms. Feedback from hundreds of users shows that TOMLAB is used for a large variety of applied optimization and mathematical programming solutions. Mostly low- and medium sized problems are solved, which is natural because up to now not many sparse solvers have been available in MATLAB. Some users of v3.0 now report solutions of very large scale problems, where the available memory is the limit. It seems that here Unix systems have an advantage because of better memory handling.

3 The Design of TOMLAB

The TOMLAB optimization environment aims at the solution of a broad range of different optimization problems, listed in Table 1. Each type of problem demands a lot of problem dependent information to be stored and distributed throughout the system. The solution is to use structure arrays for all information storage. One array, often called *Prob*, stores all information about the problem to be solved and all solver parameters, and one array, often called *Result*, stores all information about the solution of the problem. The array fields are defined in a very general way enabling all types of problems to fit into the general structure as described in [19]. This approach makes it very easy to change parameters and redesign only a small part of the system. The three User's Guides [27, 22, 21] show how the structure definition has evolved over time. In the TOMLAB system only the pointers to these structures are necessary to pass around. All user defined routines (for nonlinear problems) may get access to the full structure by adding an extra argument to the input parameter list.

A common problem when solving applied optimization problems is how to pass all problem variable information

Table 1: The different types of optimization problems treated in TOMLAB.

probType	Number	Description of the type of problem
uc	1	Unconstrained optimization (simple bounds).
qp	2	Quadratic programming.
con	3	Constrained nonlinear optimization.
ls	4	Nonlinear least squares problems.
lls	5	Linear least squares problems.
cls	6	Constrained nonlinear least squares problems.
mip	7	Mixed integer programming.
lp	8	Linear programming.
glb	9	Box-bounded global optimization.
glc	10	Box-bounded global mixed-integer optimization.
exp	11	Exponential fitting problems.
nts	12	Nonlinear time series.

around in the system, besides the decision parameters to be optimized. In the MATLAB system, one way is to use global variables, which could get rather messy, and error prone. The recommended solution in TOMLAB is to put all information as arbitrarily unused subfields in the problem structure, e.g., as subfields to *Prob.user*. By this way, information needed to evaluate the low level routines is easily retrieved. If information is computed in the low-level routines, dependent on the particular decision variables, global variables have to be used to send this information between different low-level routines. TOMLAB has a standardized way to do this. If the user adheres to these conventions, recursive calls are also possible in the system without the global variables giving rise to unpredictable errors.

The final optimization results are sent back to the calling routine using the Result structure. As usual, there is no general way to pass problem dependent information back through the solvers. Such information has to be passed using global variables or files. TOMLAB is passing back all information necessary to do warm starts of the solvers.

All TOMLAB MATLAB solvers, listed in Table 2, are written to use the input and output structure arrays. The interfaced external solvers are all Fortran solvers, see Table 3, and each connected to MATLAB with a specially written MEX-file interface. Each external solver also needs an interface routine that unpacks the structure array *Prob*, makes the call to the solver, and then assigns the appropriate fields in the structure *Result*. Using that approach the system can be designed independent on any particular solver, and can be written very general. This idea is central in implementing the key idea of TOMLAB, which is to let the user define the problem once, and then be able to run any suitable type of solver on the problem. This is a fundamental demand in a standard modeling system, but not obvious in a general environment like MATLAB. The use of structure arrays, and arrays of structure arrays, in an easy way collects all optimization results and makes advanced result presentation and statistics possible. One difference between standard modeling systems and TOMLAB is that TOMLAB has a more systematic way of returning all available solver information, which makes it suitable for algorithm development and testing as well.

Another idea in TOMLAB is to provide different types of interfaces for the optimization solution, giving the user the choice to pick the most suitable for the task at hand. The user may use a graphical user interface (GUI), a menu system, a multi-solver driver routine, or the user can directly call one of the many solvers. A flow-sheet of the process of optimization in TOMLAB is shown in Figure 1.

Applied users most often want to quickly setup the problem and get a first solution to their problem, trying out different solvers on their particular problem. This is symbolized with the *Applied user* box in the figure, which runs the *Optimization driver*. In a systematic testing phase the GUI and menu systems are valuable. In the last stage, often production runs, direct calls to the solvers give the fastest solution. The *Interface routines* in Figure 1 are used to convert computational results to the form expected by different solvers. Depending on which solver is used, different *Interface routines* are used to communicate with the *Low level routines* via the *Gateway routines*. The Gateway routines do the bookkeeping, keep track of search directions (for low-dimensional plotting), and determine type of differentiation; analytic, automatic, or any of the five types of numerical differentiation.

When using a standard modeling language, the language in itself has little interaction with the solver. The user is supposed to read the solvers user's guide, and write a text file with commands to the solver. The approach is different in TOMLAB. Still it is possible to use text files, but using the GUI makes the solver parameter handling

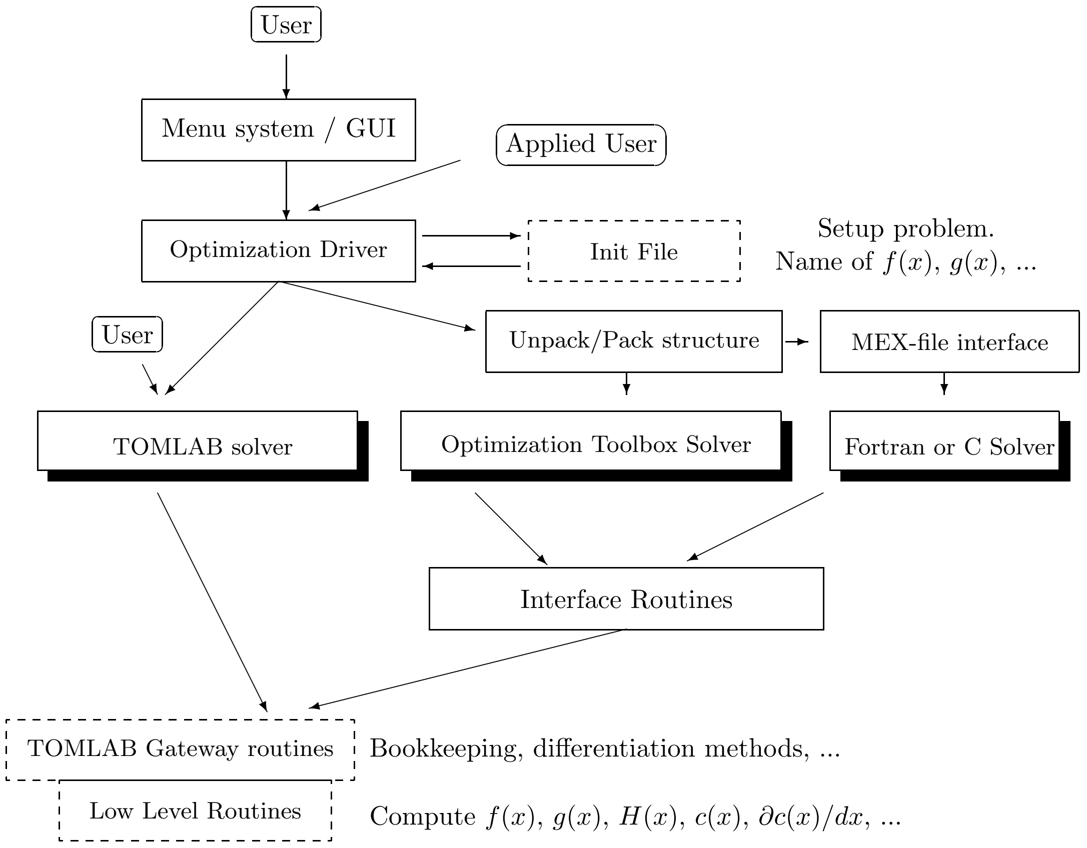


Figure 1: The process of optimization in TOMLAB.

Table 2: The MATLAB optimization solvers in TOMLAB.

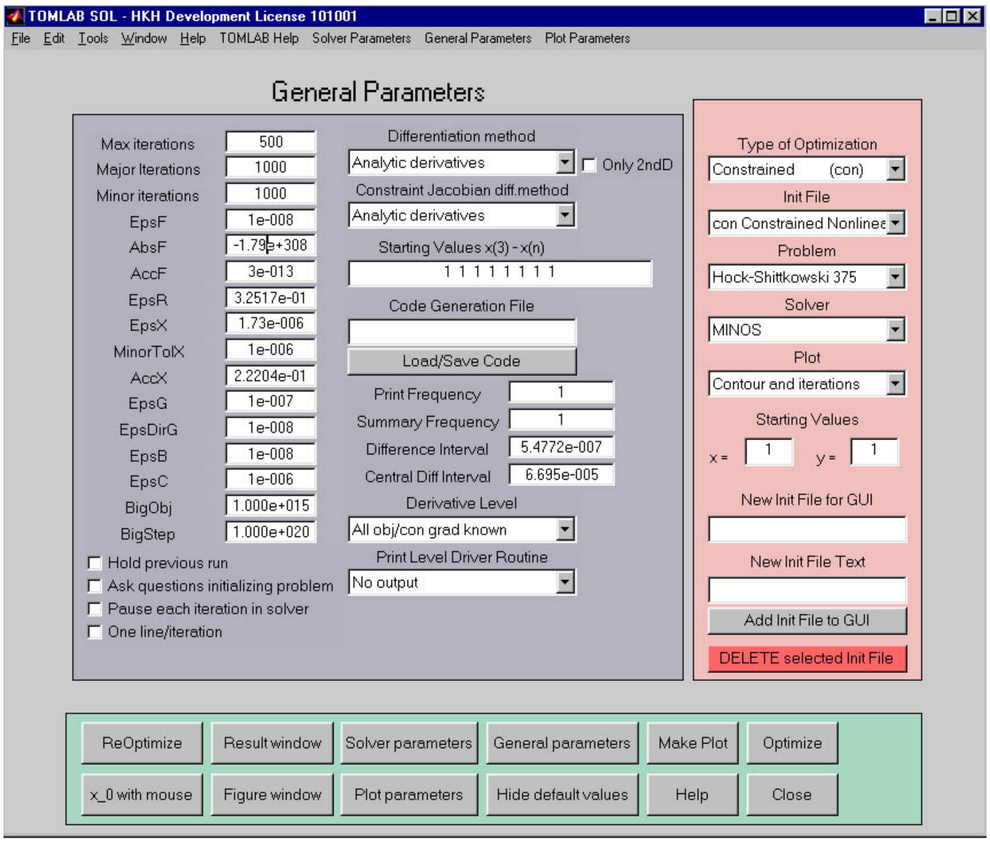
Solver	Type	Description
ucSolve	uc	Implements the Newton method, the quasi-Newton BFGS and inverse BFGS method, the quasi-Newton DFP and inverse DFP method, the Fletcher-Reeves and Polak-Ribiere conjugate-gradient method, and the Fletcher conjugate descent method. Subspace minimization techniques [33].
sTrustr	con	Convex optimization of partially separable functions, using a structural trust region algorithm [6].
glbSolve	glb	The DIRECT algorithm [30]. Fast Fortran version *glbFast*.
glcSolve	glc	Extended DIRECT [31]. Also fast Fortran version *glcFast*.
ego	glb	The Efficient Global Optimization (EGO) algorithm [32].
llsSolve	lls	Linear least squares using an active-set strategy.
clsSolve	cls	Active-set strategy [5]. Implements the Gauss-Newton method, the Al-Baali-Fletcher [3] and the Fletcher-Xu [8] hybrid method, and the Huschens TSSM method [28]. Subspace minimization handles rank problems.
conSolve	con	Sequential quadratic programming (SQP) method by Schittkowski [39].
nlpSolve	con	Filter SQP algorithm by Fletcher and Leyffer [9].
lpSolve	lp	Numerically stable revised simplex algorithm.
DualSolve	lp	Dual simplex algorithm.
qpSolve	qp	Active-set method. For indefinite programs, using directions of negative curvature to find a local minimum.
mipSolve	mip	Branch-and-bound algorithm [36, chap. 8].
cutplane	mip	Cutting plane algorithm using Gomorov cuts.

much easier and more flexible. The GUI has two modes that display solver parameters, the General Parameter mode and the Solver Parameter mode. Parameters used by most solvers are collected in the General Parameter mode. Figure 2 shows the general parameters for the case of constrained nonlinear programming when MINOS is the selected solver. The Solver Parameter window for MINOS is shown in Figure 3. In both figures the option to display the default values of the solver is selected. The drag menus give help on all solver parameters for all solvers and for the general parameters. Note that less parameters are displayed in the windows when linear or quadratic programming is the Type of Optimization, making it easier for the user to change the parameters relevant for the problem to be solved.

The GUI is also a code generator, creating one M-file and one binary MAT-file which runs the selected problem and solver, with all selections made for the particular problem [7]. As seen in Figure 2 the user gives the name of the *Code Generation File*, and may also load the information back into the GUI at a later session enabling the user to quickly change any settings. The generated code is directly executable from the command line and easy to generalize, e.g., to solve a whole set of problems with the particular solver settings.

There are two ways to create a TOMLAB problem structure and solve the optimization problem, either using the *TOMLAB Quick* format (*TQ* format) or the *Init File* format (*IF* format). The *TQ* format is a quick way to setup a problem and easily solve it using any of the TOMLAB solvers. The solution process using the *TQ* format has four steps:

1. Define the problem structure Prob and any user supplied function routines.



General Parameters

Max iterations	500
Major Iterations	1000
Minor iterations	1000
EpsF	1e-008
AbsF	-1.79e+308
AccF	3e-013
EpsR	3.2517e-01
EpsX	1.73e-006
MinorTolX	1e-006
AccX	2.2204e-01
EpsG	1e-007
EpsDirG	1e-008
EpsB	1e-008
EpsC	1e-006
BigObj	1.000e+015
BigStep	1.000e+020

☐ Hold previous run
☐ Ask questions initializing problem
☐ Pause each iteration in solver
☐ One line/iteration

Differentiation method	Analytic derivatives ☐ Only 2ndD
Constraint Jacobian diff.method	Analytic derivatives
Starting Values x(3) - x(n)	1 1 1 1 1 1 1 1
Code Generation File	
Print Frequency	1
Summary Frequency	1
Difference Interval	5.4772e-007
Central Diff Interval	6.695e-005
Derivative Level	All obj/con grad known
Print Level Driver Routine	No output

Type of Optimization	Constrained (con)
Init File	con Constrained Nonline
Problem	Hock-Shittkowski 375
Solver	MINOS
Plot	Contour and iterations
Starting Values	x = 1 y = 1
New Init File for GUI	
New Init File Text	

Figure 2: The GUI in General Parameter Mode solving constrained nonlinear problems.

2. Make necessary changes in the Prob structure.
3. Call the solver or the multi-solver driver routine.
4. Perform postprocessing, e.g., print the result of the optimization.

Step 1 could be done in several ways in TOMLAB. Recommended is to call one of the *assign* routines dependending on the type of optimization problem.

MATLAB call	**probType**	**Type of optimization problem**
Prob = qpAssign(...)	2	Quadratic programming.
Prob = conAssign(...)	1,3	Nonlinear optimization.
Prob = clsAssign(...)	4,5,6	Linear and nonlinear least squares.
Prob = mipAssign(...)	7	Mixed-integer programming.
Prob = lpAssign(...)	8	Linear programming.
Prob = glcAssign(...)	9,10	Global optimization.

Step 3 is either a direct call to the solver, e.g.,

```
Result = conSolve(Prob);
```

or a call to the multi-solver driver routine *tomRun*, e.g.,

```
Result = tomRun('conSolve', Prob);
```

When solving a sequence a similar problems, the best way is to create the problem structure once, then make a loop, do the changes in the structure, and solve the problem for each change. As a complete example, the solution of a constrained optimization problem is shown. The function is an exponential times a polynomial. There are

Table 3: MEX-file interfaces in TOMLAB (partial list).
An S in the S/D column means a sparse solver, and D a dense solver.

Solver Name	Type	S/D	Developers
MINOS 5.51	**con**	S	B. Murtagh and M.A. Saunders.
SNOPT 6.1-1	**con**	S	P.E. Gill, W. Murray and M.A. Saunders.
NPSOL 5.02	**con**	D	P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright.
NPOPT 1.0-10	**con**	D	P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright.
FFSQP 3.7-b	**con**	D	J.L. Zhou, A.L. Tits and C.T. Lawrence.
SQOPT 5.3-5	**qp**	S	P.E. Gill, W. Murray and M.A. Saunders.
QPOPT 1.0-10	**qp**	D	P.E. Gill, W. Murray and M.A. Saunders.
LPOPT 1.0-10	**lp**	D	P.E. Gill, W. Murray and M.A. Saunders.
NLSSOL 5.0-2	**cls**	D	P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright.
LSSOL 1.05-4	**lls**	D	P.E. Gill, S.J. Hammarling, W. Murray, M.A. Saunders and M.H. Wright.
LSQR	**lls**	S	C.C Paige and M.A. Saunders.
XPRESS-MP	**mip**	S	Dash Optimization Ltd.
BTNCLC 1.0	**con**	D	H Schramm. (Nonsmooth $f(x)$, linear constraints)

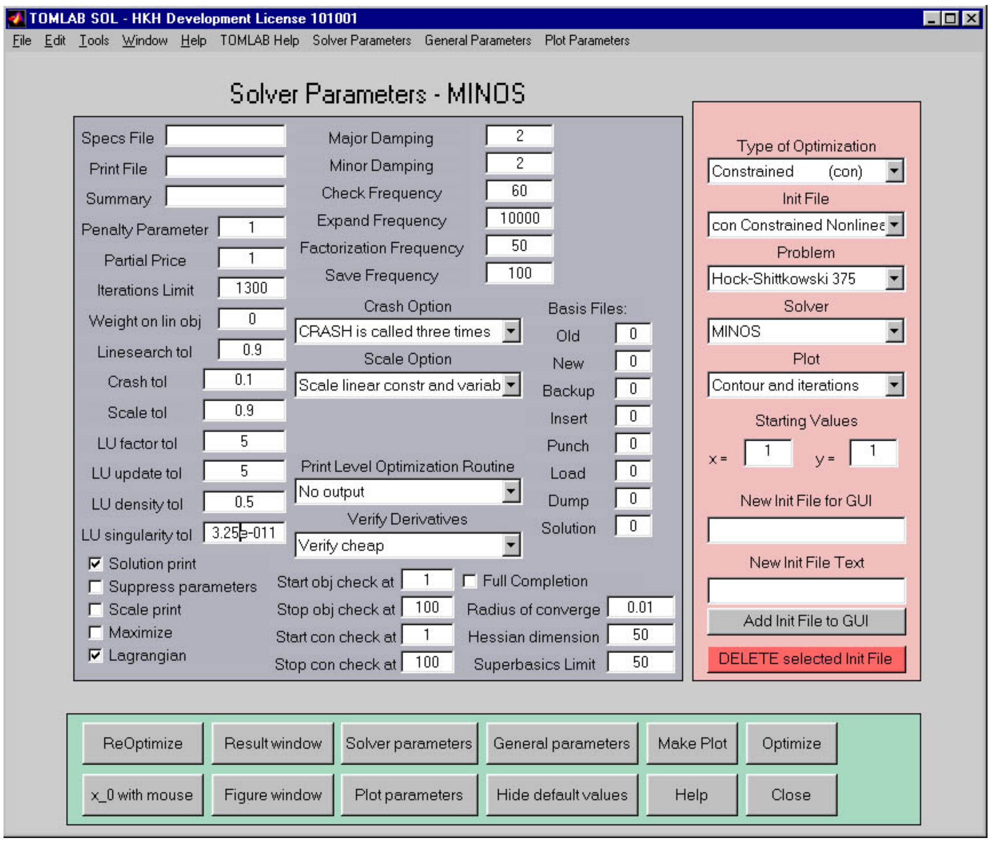
two nonlinear (quadratic) inequality constraints, and one linear equality constraint. The problem is solved one hundred times for different starting values in the interval $[-100, 100]$. The actual function, gradient and Hessian routine are shown in Appendix A and the nonlinear constraints and the constraint gradient routine in Appendix B.

```
Name    = 'Constrained exponential problem';
A       = [1 1];        % One linear constraint
b_L     = 0;            % Lower bound on linear constraint
b_U     = 0;            % b_L == b_U implies equality
c_L     = zeros(2,1);   % Nonlinear inequality constraints, lower bound 0
c_U     = [];           % Empty means Inf (default)
x_0     = [-5;5];       % Initial value
x_L     = [-10;-10];    % Lower bounds on x
x_U     = [10;10];      % Upper bounds on x
fLowBnd = 0;            % A lower bound on the optimal function value

% HessPattern is a sparse matrix giving the nonzero pattern in the Hessian
HessPattern = [];  % All elements in Hessian are nonzero.
% ConsPattern, pattern of nonzeros in the constraint gradient matrix
ConsPattern = [];  % All elements in the constraint Jacobian are nonzero.
pSepFunc    = [];  % The function f is not defined as separable

% Generate the problem structure using the TOMLAB Quick format
% Routines con_f, con_g, con_H, con_c and con_dc are used
Prob = conAssign('con1_f', 'con1_g', 'con1_H', HessPattern, x_L, x_U, ...
                 Name, x_0, pSepFunc, fLowBnd, A, b_L, b_U, ...
                 'con1_c', 'con1_dc', [], ConsPattern, c_L, c_U);
for i=1:100
    Prob.x_0 = -100 + 200*rand(2,1);  % Random initial values for x
    Result   = tomRun('NPSOL', Prob); % Solve problem with NPSOL
    PrintResult(Result);              % Print result
    f_k(i)   = Result.f_k;            % Save optimal f(x)
    x_k(:,i) = Result.x_k;            % Save x for postprocessing
end
```

The more complicated *IF* format makes it possible to use the GUI and menu system as well. In the *IF* format one initialization file is defined for each set of user problems. The set could consist of only one problem. The


TOMLAB SOL - HKH Development License 101001
File Edit Tools Window Help TOMLAB Help Solver Parameters General Parameters Plot Parameters
Solver Parameters - MINOS
ReOptimize
Result window
Solver parameters
General parameters
Make Plot
Optimize
x_0 with mouse
Figure window
Plot parameters
Hide default values
Help
Close
Add Init File to GUI
DELETE selected Init File

Specs File
Print File
Summary
Penalty Parameter 1
Partial Price 1
Iterations Limit 1300
Weight on lin obj 0
Linesearch tol 0.9
Crash tol 0.1
Scale tol 0.9
LU factor tol 5
LU update tol 5
LU density tol 0.5
LU singularity tol 3.25e-011
☑ Solution print
☐ Suppress parameters
☐ Scale print
☐ Maximize
☑ Lagrangian

Major Damping 2
Minor Damping 2
Check Frequency 60
Expand Frequency 10000
Factorization Frequency 50
Save Frequency 100
Crash Option: CRASH is called three times
Scale Option: Scale linear constr and variab
Print Level Optimization Routine: No output
Verify Derivatives: Verify cheap

Basis Files:
Old 0
New 0
Backup 0
Insert 0
Punch 0
Load 0
Dump 0
Solution 0

Start obj check at 1
Stop obj check at 100
Start con check at 1
Stop con check at 100
☐ Full Completion
Radius of converge 0.01
Hessian dimension 50
Superbasics Limit 50

Type of Optimization: Constrained (con)
Init File: con Constrained Nonlinea
Problem: Hock-Shittkowski 375
Solver: MINOS
Plot: Contour and iterations
Starting Values x = 1 y = 1
New Init File for GUI
New Init File Text

Figure 3: The Solver Parameters for the MINOS solver.

initialization file should perform four tasks:

- If the input problem number is empty, return a string matrix with the *i*:th row defining the name of the *i*:th problem defined in the file.
- If the input problem number is nonempty, return the TOMLAB problem structure corresponding to this problem number.
- If a flag is set, ask user defined questions, otherwise use default values and do a quiet setup. Most often a problem dimension is asked for, or a choice of a certain data series. TOMLAB includes query routines.
- If an input problem structure is given, override the default values with the values set in the input structure.

The exponential problem discussed above is defined in the *IF* format as problem three in **con_prob.m** in the TOMLAB distribution. The following example shows how to solve the problem in this case, making two changes in the default problem definition by setting two fields in the problem structure beforehand:

```
Prob.optParam.MaxIter = 1000;  % Increase maximal numbers of iterations
Prob.x_0 = [0 1 3]';           % Change default initial value of x
% tomRun solves problem 3 defined in con_prob using solver conSolve.
Result = tomRun('conSolve', 'con_prob', 3, Prob);
```

The TOMLAB User's Guide describes how to create and edit new *Init Files*. Most users define their problem with the *TQ* format and want a quick way to access their problems in the GUI. If dropping the wish to ask user questions, then a simpler type of initialization file may be created. Using three routines (*newInitFile*, *addProb* and *makeInitFile*), it is easy to collect a group of any number of problem structures Prob into a set and save them in the *IF* format. To benefit from the GUI design, the problems in the same *Init File* should be of the same optimization problem type, e.g., quadratic programs. The Prob structures are saved in a binary MAT-file having the same name as the *Init File*. Having defined a proper file in the *IF* format, it is added to the GUI calling the routine *AddProblemFile*. *Init Files* could also be added and deleted directly in the GUI as seen at the right side of Figure 2.

When solving a sequence a similar problems, the best way is to pick up the problem structure once using *probInit*, and then make a loop, do the changes in the structure, and solve the problem for each change. This means that using *makeInitFile* to quickly make an *Init File* is not much of a restriction, because any user questions or changes in the structure could be made after the creation, but before the call to the solver. The following example shows how the same exponential problem with random initial values as above is solved using the *IF* format.

```
probFile   = 'con_prob';   % Name of the Init File
probNumber = 3;            % Problem number in the Init File
ask        = [];           % Ask flag not used
Prob       = [];           % No default values changed

Prob = probInit(probFile, probNumber, ask, Prob)

for i=1:100
    Prob.x_0 = -100 + 200*rand(2,1);
    Result   = tomRun('conSolve', Prob);
    f_k(i)   = Result.f_k;  % Save optimal f(x)
    x_k(:,i) = Result.x_k;  % Save optimal x for postprocessing
end
```

Note that for LP, QP and MIP problems there is another alternative that creates a file in the *IF* format. The routines *lpAssign*, *qpAssign* and *mipAssign* have an option to create *Init Files* with an arbitrarily number of problems, see the last example in Section 4. With this strategy the problems are saved more efficiently with regards to space. One binary MAT-file is created for each problem in the *Init File*. If the number of problems are large, this alternative may be preferable.

4 Linear, Quadratic and Discrete Optimization

Linear, quadratic and mixed-integer programming, as well as linear least squares, have the common feature that the full problem is possible to formulate before calling the solver, and no user function routines are needed. This makes it possible to create *Init Files* for large sets of problems easily, as seen in the last example in this section. The formulation in TOMLAB for a linear program (**lp**) is

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

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b_L, b_U \in \mathbb{R}^m$. Consider the solution of a very simple linear programming problem

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

named *lpExample*. Define the following statements in *lpExample.m*

```
Name  = 'lpExample';
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
b_L   = []; x_U = []; x_0 = [];
```

The simplest way to solve the problem in TOMLAB using the *TQ* format, is to define the Matlab statements

```
lpExample;
Prob   = lpAssign(c, A, b_L, b_U, x_L, x_U, x_0, Name);
Result = lpSolve(Prob);
PrintResult(Result);
```

The call to *lpExample* defines the problem. Because of the structure approach, two more statements are needed instead of a direct call to the solver. lpAssign creates the problem structure Prob, which is then used as input to the solver. For LP problems, as well as QP and MIP problems, the assign routine is also a M-file code generator able to automatically create files in the *IF* format. Using the same example (2) to illustrate this feature gives

```
lpExample;
lpAssign(c, A, b_L, b_U, x_L, x_U, x_0, Name, 'lptest', 1);
AddProblemFile('lptest','One simple LP test problem','lp');
tomRun('lpSolve','lptest',1);
```

Two more parameters are needed in the call to *lpAssign.* The first tells *lpAssign* to define a new *Init File* called *lptest.* The second extra parameter, now set as one, is called *nProblem* and is the number of problems to be defined. It determines which of two types of *Init Files* is created. The *lptest* file is included in the GUI Init File data base by the call to *AddProblemFile*, and the multi-solver driver routine *tomRun* calls the same solver as in the previous example. Thus, the result will be exactly the same as for the first example. When setting *nProblem* as empty or one, as in this case, *lpAssign* defines a program structure in the text file **lptest.m** in which the user easily can insert more test problems. Comments in the created file guides the user in how to define a new problem. A new problem definition can be loaded by execution of a script, by reading from a stored file or the user can explicitly write the Matlab statements that define the new problem in the file.

It is easy to create an *Init File* for a large set of test problems, using the second type of *Init File*, created if *nProblem* > 1. This feature is illustrated by running a test where the coefficients in the objective function in the previous test problem are disturbed by random noise. For our example, restrict the large number of test problems to be three.

```
LargeNumber=3;
lpExample;
n = length(c);
for i=1:LargeNumber
    cNew = c + 0.1*(rand(n,1)-0.5); % Add random disturbances to vector c
    if i == 1
       % lpAssign defines lplarge.m for LargeNumber testproblems
       % and stores the first problem in lplarge_P1.mat
       k = [1,LargeNumber];
    else
       k = i; % The i:th problem is stored in lplarge_Pi.mat
    end
    lpAssign(cNew, A, b_L, b_U, x_L, x_U, x_0, Name,'lplarge', k);
end
% Add lplarge to the GUI Init File data base as an LP problem
AddProblemFile('lplarge','Large set of randomized LP problems','lp');
runtest('lpSolve',0,'lplarge',1:LargeNumber);
```

Each problem defined gets a unique name. In the first iteration, $i = 1$, *lpAssign* creates an *Init File* defined for three test problems, and saves the first test problem information in *lplarge_P1.mat.* In the other iterations *lpAssign* just defines the other MAT-files, *lplarge_P2.mat* and *lplarge_P3.mat.* The results running the *runtest* utility gives

```
Solver: lpSolve.
Problem  1: LPTest - 1             f_k     -26.501771581492278000
Problem  2: LPTest - 2             f_k     -26.546357769596234000
Problem  3: LPTest - 3             f_k     -26.425877951614158000
```

To illustrate how MATLAB may be used to build mathematical models, the conversion of a general assignment problem (GAP) to a problem formulation suitable for a MIP solver is shown. The GAP problem is formulated as

$$
\begin{array}{ll}
\min\limits_{x} & f(x) = \sum_i^m \sum_j^n c_{i,j} \cdot x_{i,j} \\
 & \\
s/t & \begin{array}{lcll} \sum_j^m a_{i,j} & \leq & b_j & \text{for} \quad j = 1, ..., n, \\ \sum_j^n x_{i,j} & = & 1 & \text{for} \quad i = 1, ..., m, \quad x \text{ binary } \ 0/1 \end{array}
\end{array}
\tag{3}
$$

The following simple, but general, MATLAB routine converts a given $m \times n$ matrix A to a sparse $(n+m) \times 2 \cdot N$ matrix to be used in the call to XPRESS-MP to solve the GAP problem.

```
function a = a2gap( A);

[m,n] = size(A); N = m*n; a = spalloc(n+m,N,N+N); k = 0;
for j = 1:n
    a(j,k+[1:m]) = A(:,j)';   % First part, the A values
    k = k+m;
end
v  = m*[0:n-1];
for j=1:m, a(n+j,j+v)=1; end  % Second part, chunks of ones
```

This routine, and similar ones, are part of the TOMLAB/XPRESS-MP toolbox. A number of callbacks are implemented in the XPRESS-MP interface. This makes it possible to implement heuristics, special cuts and other options in MATLAB code as part of the MIP solution process. In each callback all problem information as well as all XPRESS-MP control and problem variables are available and possible to change. Manual user control is also possible.

TOMLAB also includes a collection of additional stand-alone tools for linear, quadratic and discrete optimization. In Table 4 some of the solvers are listed. The implementation of the network programming algorithms are based on the forward and reverse star representation technique described in Ahuja et al. [2, pages 35-36]. For a full list and details, see the TOMLAB User's Guide [23].

Table 4: Some of the linear and discrete (LDO) solvers in TOMLAB v3.0.

Function	**Description**
dijkstra	Shortest path solver using Dijkstra's algorithm [1, pages 250-251].
modlabel	Shortest path using a modified label correcting method [1, page 262].
TPsimplx	Transportation simplex algorithm [34, chap 5.4].
NWsimplx	Minimum cost network flow simplex algorithm [2, page 415].
salesman	Symmetric traveling salesman solver using Lagrangian relaxation and the subgradient method with the Polyak rule II [11].
balas	Branch and bound algorithm for binary IP using Balas method.
dpknap	Forward recursion dynamic programming for knapsack problems.
cutstock	Cutting stock algorithm using revised simplex and *dpknap*.

5 Nonlinear Parameter Estimation and Nonlinear Least Squares

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

$$\begin{array}{cccccc} \min\limits_{x} & f(x) = \frac{1}{2} r(x)^T r(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U, \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{4}$$

where $x, x_L, x_U \in \mathbb{R}^n$, $r(x) \in \mathbb{R}^N$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$. The TOMLAB nonlinear least squares solver *clsSolve* is used in several of our research projects, e.g., estimation of formation constants in chemical equilibrium analysis, analysis of plasmid stability in fermentation processes and fitting of exponential sums to empirical data in radiotherapy planning. Examples of some exponential sum model fitting problems will be given in Section 6.

A standard test set for nonlinear squares problems is the test set of More, Garbow and Hillstrom [35] with 35 problems. A summary of a comparison with different solvers is shown in in Table 5. The TOMLAB *clsSolve* with the Fletcher-Xu hybrid method (FX) has no failures, and is faster than any other solver in the test. *clsSolve* with Gauss-Newton fails three times and need more evaluations to converge. *NLSSOL* has three failures, but needs very many evaluations to converge. The Optimization Toolbox 2.0 routine *lsqnonlin* and *NPOPT* are not robust on these problems.

Table 5: Comparison of algorithmic performance for the More, Garbow, Hillstrom test set for nonlinear least squares problems. The test set has 35 problems. A summary with number of failures and the median number of iterations, residual evaluations and Jacobian evaluations are shown for the four solvers. Two algorithms from the Math Works Optimization Toolbox 2.0 *lsqnonlin* solver are tested, a Levenberg-Marquardt algorithm (*lsq LM*) and a large scale trust-region algorithm (*lsq LS*).

	clsSolve GN	clsSolve FX	lsq LM	lsq LS	NLSSOL	NPOPT
Median	9 11.5 11.5	9 12 12	16 17 17	13 13 13	15 25.5 25.5	22 35 35
Failures	3	0	6	13	3	8

The results are similar when adding simple bounds on eighteen of the problems in the More, Garbow and Hillstrom test set, as suggested by David Gay. Summary results are shown in Table 5. Once again the TOMLAB *clsSolve* FX is the fastest and most reliable solver. Full test results for both test sets are found in [20].

Table 6: Comparison of algorithmic performance for the More, Garbow, Hillstrom test set for nonlinear least squares problems, with additional simple bounds by D. Gay. The test set consists of 18 problems. A summary with number of failures and the median number of iterations, residual evaluations and Jacobian evaluations are shown for the four solvers.

	clsSolve GN	clsSolve FX	lsqnonlin LS	NLSSOL	NPOPT
Median	9.5 11.5 11.5	8 9 9	23 23 23	9 11 11	17 24 24
Failures	2	1	6	2	1

A comparison of *clsSolve* and other solvers on 26 linearly constrained nonlinear least squares test problems are shown in Table 7. The TOMLAB *clsSolve* FX and GN, and NLSSOL, are the fastest and most reliable solvers in this test. The Optimization Toolbox 2.0 routine *lsqnonlin* does not solve constrained problems. Full test results are found in [20]. For more details on the test problems, and tests on TOMLAB v1.0 solvers, see the thesis of Björkman [5].

Table 7: A summary of a comparison of algorithmic performance for linearly constrained nonlinear least squares problems.

	clsSolve GN	clsSolve FX	clsSolve Huschen	NLSSOL	NPOPT
Average	6 8 8	6 8 7	23 24 24	6 9 9	12 15 15
Failures	0	0	0	1	0

6 Parameter Estimation in Exponential Models

In TOMLAB the problem of fitting sums of positively weighted exponential functions to empirical data may be formulated either as a nonlinear least squares problem or a separable nonlinear least squares problem [38]. Several empirical data series are predefined and artificial data series may also be generated. Algorithms to find starting values for different number of exponential terms are implemented. Test results show that these initial value algorithms are very close to the true solution for equidistant problems and fairly good for non-equidistant problems, see the thesis by Petersson [37]. Good initial values are extremely important when solving real life exponential fitting problems, because they are so ill-conditioned.

There are five different types of exponential models with special treatment in TOMLAB, shown in Table 8. In research in cooperation with Todd Walton, Vicksburg, USA, TOMLAB has been used to estimate parameters using maximum likelihood in simulated Weibull distributions, and Gumbel and Gamma distributions with real data. TOMLAB has also been useful for parameter estimation in stochastic hydrology using real-life data.

Table 8: Exponential models treated in TOMLAB.

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

In [24] algorithms for fitting exponential sums $D(r) = \sum_{i=1}^p a_i(1 - \exp(-b_i r))$ to numerical data are presented and compared for efficiency and robustness. The numerical examples stem from parameter estimation in dose calculation for radiotherapy planning. The doses are simulated by emitting an ionizing photon beam into water and at different depths d and different radius r from the beam center measuring the absorption. The absorbed dose is normally distinguished into primary dose from particles excited by the photon beam and scattered dose from the following particle interactions. In Table 6 summary results are presented from a comparison between the same solvers as in the first test in Section 5. Tests are performed on 335 radiotherapy problems using starting values obtained by the TOMLAB initial value algorithm. Note that the lsqnonlin LS algorithm has convergence problems, giving many failures. The efficiency of the TOMLAB v1.0 *clsSolve* Fletcher-Xu method with the separable algorithm is obvious, less than 65 percent of the number of iterations, residual evaluations and Jacobian evaluations, compared to the best of the other solvers, are required to solve the problems. A more detailed result table is given in [25].

7 Global Optimization

The solver *glbSolve* implements an algorithm for **box-bounded global optimization (glb)**, i.e. problems that have finite simple bounds on all the variables,

Table 9: Numerical test on the nonlinear least squares solution of 335 problems from radiotherapy planning supplied by Helax AB, Uppsala, Sweden. A summary with number of failures and the mean number of iterations, residual evaluations and Jacobian evaluations are shown for the four solvers. Two algorithms from the Math Works Optimization Toolbox 2.0 *lsqnonlin* solver are tested, a Levenberg-Marquardt algorithm (*lsq LM*) and a large scale trust-region algorithm (*lsq LS*). clsSolve FX† runs a separable nonlinear least squares algorithm, Algorithm II in [38].

Depth d	clsSolve GN	clsSolve FX†	lsq LM	lsq LS	NLSSOL	NPOPT
Average	16 18 18	8 10 10	19 20 20	25 25 25	13 18 18	21 31 31
Failures	0	0	0	68	0	0

$$\begin{array}{ll} \min\limits_{x} & f(x) \\ & \\ s/t & -\infty < \quad x_L \quad \leq \quad x \quad \leq \quad x_U \quad < \infty, \end{array} \tag{5}$$

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

glbSolve implements the DIRECT algorithm [30], which is a modification of the standard Lipschitzian approach that eliminates the need to specify a Lipschitz constant. In *glbSolve* no derivative information is used. A new much faster Fortran version of *glbSolve* is available as the solver *glbFast*, called using a MEX-file interface. For global optimization problems with expensive function evaluations the routine *ego* implements the Efficient Global Optimization (EGO) algorithm [32]. The idea of the EGO algorithm is to first fit a response surface to data collected by evaluating the objective function at a few points. Then, EGO balances between finding the minimum of the surface and improving the approximation by sampling where the prediction error may be high.

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

$$\begin{array}{llcccccl} \min\limits_{x} & f(x) & & & & & & \\ & & & & & & & \\ & -\infty < & x_L & \leq & x & \leq & x_U & < \infty \\ s/t & & b_L & \leq & Ax & \leq & b_U & \\ & & c_L & \leq & c(x) & \leq & c_U, & x_j \in \mathbb{N} \quad \forall j \in \mathrm{I}, \end{array} \tag{6}$$

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 solver *glcSolve* implements an extended version of the DIRECT algorithm [31], that handles problems with both nonlinear and integer constraints. A new much faster Fortran version of *glcSolve* is available as the solver *glcFast*, called using a MEX-file interface.

The global optimization solvers may be used to find initial values for nonlinear optimization. An interesting application from design of robust controllers is presented in [10], where *glbSolve*, *glcSolve*, *MINOS*, *SNOPT* used together are able to solve the non-convex multi-stage optimization problem formulated.

The two-dimensional test function *Shekel's foxholes* from the First International Contest on Evolutionary Optimization (ICEO) is illustrative to show how effective *glbSolve* could be on problems with several local (non-global) minima. A contour plot of the function, with dots indicating points where the function value has been computed, is shown in Figure 4. After 21 iterations and 147 function evaluations, the best function value found by *glbSolve* is -12.119 at $x = (8.0239, 9.1467)^T$. As can be seen in Figure 4, most of the sampled points are concentrated in the area around the global minimum up in the right corner of the bounding box and very few function evaluations are wasted on sampling around the many local minima.

8 Applications of TOMLAB

TOMLAB has been used in a wide range of applied mathematical projects. Here we give a few examples of its successful use.

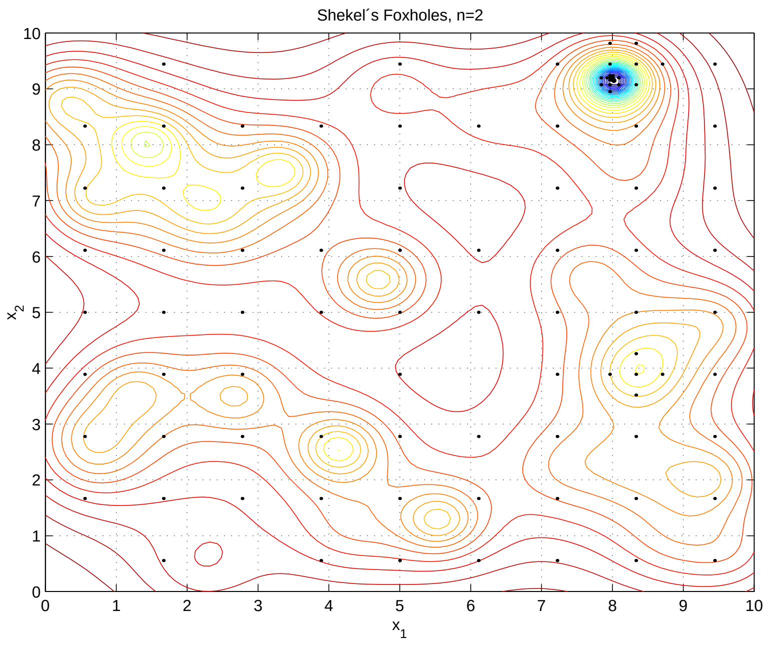


Figure 4: Contour plot of the Shekels's Foxholes function with the 147 sampled points needed for convergence using *glbSolve*.

To find unknown species formation constants and other unknown parameters in multi-phase chemical equilibrium analysis, a separable nonlinear least squares algorithm is formulated. The separation aims at using the structure of a large, ill-conditioned parameter estimation problem, to obtain a solvable set of optimization problems. Each iteration in the separable algorithm consists of a major problem and the solution of hundreds or thousands of minor ill-conditioned problems. To obtain a practically useful tool, a combination of approximation algorithms to find initial values for the unknown parameters is needed, together with robust constrained nonlinear least squares solvers [17]. As we are using the weighted L_2-norm, sensitive to outliers, all the minor problems must be solved with high accuracy and no failures. Here the TOMLAB constrained nonlinear least squares solver *clsSolve* is used in our new Matlab toolbox LAKE TB. Tests show that *clsSolve* converges faster and gives better accuracy than the previous solver, that is part of our Fortran program package LAKE, used in inorganic chemical research since 1987 [29]. Preliminary results are discussed in [17]. As TOMLAB handles recursive calls in a proper way, it is possible to use *clsSolve* for both the major and minor optimization problems.

In a joint project with Jordan M. Berg, Texas Tech University, Lubbock, TX, we want to find accurate low-order phenomenological models of thin film etching and deposition processes. These processes are central to the manufacture of micro-electronic devices. We have developed algorithms and software for parameter estimation using level sets. i.e. the problem of selecting the member of a parameterized family of curves that best matches a given curve. Level set methods offer several attractive features for treating such problems. The method is completely geometric; there is no need to introduce an arbitrary coordinate system for the curves. We have derived analytic results necessary for the application of gradient descent algorithms, and made numerical computations using TOMLAB [4].

In our research on prediction methods in computational finance, we study the prediction of various kinds of quantities related to stock markets, like stock prices, stock volatility and ranking measures. In one project we instead of the classical time series approach used the more realistic prediction problem of building a multi-stock artificial trader (ASTA). The behavior of the trader is controlled by a parameter vector which is tuned for best performance. The global optimization routine *glbSolve* is used to find the optimal parameters for the noisy functions obtained, when running on a large database of Swedish stock market data [13, 14].

9 Conclusions

We have shown that using the high-level features of MATLAB, the TOMLAB optimization environment gives a uniform front-end for the solution of most types of optimization problems. In some cases TOMLAB also acts as a code generator. The graphical user interface is a simple and powerful way to change all the solver settings and other options used for the optimization. Having access to the full power of MATLAB in the callbacks during the MIP solution process makes advanced problem dependent callback strategies possible to develop.

Acknowledgements

Numerous users, students and researchers world-wide have contributed with bits and pieces, which I am truly grateful for. Without all this help, the development of the TOMLAB optimization environment would not have been possible.

In making the latest version of TOMLAB, I want to mention Michael Saunders, who is a constant source of inspiration and assisted in making the MEX-file interfaces for the SOL solvers. Also the help provided by Hans and Renate Mittelmann have been crucial in making TOMLAB work for HP, Linux and SGI systems.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*. Elsevier/North Holland, Amsterdam, The Netherlands, 1989.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall Inc., Kanpur and Cambridge, 1993.

[3] M. Al-Baali and R. Fletcher. Variational methods for non-linear least squares. *J. Oper. Res. Soc.*, 36:405–421, 1985.

[4] Jordan M. Berg and K. Holmström. On Parameter Estimation Using Level Sets. *SIAM Journal on Control and Optimization*, 37(5):1372–1393, 1999.

[5] M. Björkman. Nonlinear Least Squares with Inequality Constraints. Bachelor Thesis, Department of Mathematics and Physics, Mälardalen University, Sweden, 1998. Supervised by K. Holmström.

[6] A. R. Conn, N. I. M. Gould, A. Sartenaer, and P. L. Toint. Convergence properties of minimization algorithms for convex constraints using a structured trust region. *SIAM Journal on Scientific and Statistical Computing*, 6(4):1059–1086, 1996.

[7] E. Dotzauer and K. Holmström. The TOMLAB Graphical User Interface for Nonlinear Programming. *Advanced Modeling and Optimization*, 1(2):9–16, 1999.

[8] R. Fletcher and C. Xu. Hybrid methods for nonlinear least squares. *IMA Journal of Numerical Analysis*, 7:371–389, 1987.

[9] Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. Technical Report NA/171, University of Dundee, 22 September 1997.

[10] C-M Fransson, B. Lennartson, and T. Wik K. Holmström. Multi criteria controller design for uncertain mimo systems using global nonconvex optimization. In *Proceedings of the 40th Conference on Decision and Control*, Orlando, Florida, 2001. Accepted.

[11] Michael Held and Richard M. Karp. The Traveling-Salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[12] T. Hellström and K. Holmström. Parameter Tuning in Trading Algorithms using ASTA. In Y. S. Abu-Mostafa, B. LeBaron, A. W. Lo, and A. S. Weigend, editors, *Computational Finance 1999*, Cambridge, MA, 1999. MIT Press.

[13] T. Hellström and K. Holmström. Parameter Tuning in Trading Algorithms using ASTA. In Y. S. Abu-Mostafa, B. LeBaron, A. W. Lo, and A. S. Weigend, editors, *Computational Finance (CF99) – Abstracts of the Sixth International Conference, Leonard N. Stern School of Business, January 1999*, Leonard N. Stern School of Business, New York University, 1999. Department of Statistics and Operations Research.

[14] T. Hellström and K. Holmström. Global Optimization of Costly Nonconvex Functions, with Financial Applications. *Theory of Stochastic Processes*, 7(23)(?):??–??, 2001. To be published.

[15] K. Holmström. TOMLAB - A General Purpose, Open Matlab Environment for Research and Teaching in Optimization. Technical Report IMa-TOM-1997-03, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997. Presented at the 16th International Symposium on Mathematical Programming, Lausanne, Switzerland, August 24-29, 1997.

[16] K. Holmström. TOMLAB - An Environment for Solving Optimization Problems in Matlab. In M. Olsson, editor, *Proceedings for the Nordic Matlab Conference '97, October 27-28*, Stockholm, Sweden, 1997. Computer Solutions Europe AB.

[17] K. Holmström. Constrained Separable NLLS Algorithms for Chemical Equilibrium Analysis. In Arne Løkketangen, editor, *Proceedings from the 5th Meeting of the Nordic Section of the Mathematical Programming Society*, ISBN 82-90347-76-6, Molde, 1998. Division of Operations Research, Molde University.

[18] K. Holmström. New Optimization Algorithms and Software. *Theory of Stochastic Processes*, 5(21)(1-2):55–63, 1999.

[19] K. Holmström. The TOMLAB Optimization Environment in Matlab. *Advanced Modeling and Optimization*, 1(1):47–69, 1999.

[20] K. Holmström. The TOMLAB v2.0 Optimization Environment. In E. Dotzauer, M. Björkman, and K. Holmström, editors, *Sixth Meeting of the Nordic Section of the Mathematical Programming Society. Proceedings*, Opuscula 49, ISSN 1400-5468, Västerås, 1999. Mälardalen University, Sweden.

[21] K. Holmström. The TOMLAB Optimization Environment User's Guide. Technical Report IMa-TOM-2000-02, Department of Mathematics and Physics, Mälardalen University, Sweden, 2000.

[22] K. Holmström. TOMLAB v2.0 User's Guide. Technical Report IMa-TOM-2000-01, Department of Mathematics and Physics, Mälardalen University, Sweden, 2000.

[23] K. Holmström. TOMLAB v3.0 User's Guide. Technical Report IMa-TOM-2001-01, Department of Mathematics and Physics, Mälardalen University, Sweden, 2001.

[24] K. Holmström, Anders Ahnesjö, and J. Petersson. Algorithms for exponential sum fitting in radiotherapy planning. Technical Report IMa-TOM-2001-02, Department of Mathematics and Physics, Mälardalen University, Sweden, 2001.

[25] K. Holmström and M. Björkman. The TOMLAB NLPLIB Toolbox for Nonlinear Programming. *Advanced Modeling and Optimization*, 1(1):70–86, 1999.

[26] K. Holmström, M. Björkman, and E. Dotzauer. The TOMLAB OPERA Toolbox for Linear and Discrete Optimization. *Advanced Modeling and Optimization*, 1(2):1–8, 1999.

[27] K. Holmström, M. Björkman, and E. Dotzauer. TOMLAB v1.0 User's Guide. Technical Report IMa-TOM-1999-01, Department of Mathematics and Physics, Mälardalen University, Sweden, 1999.

[28] J. Huschens. On the use of product structure in secant methods for nonlinear least squares problems. *SIAM Journal on Optimization*, 4(1):108–129, 1994.

[29] N. Ingri, I. Andersson, L. Pettersson, L. Andersson, A. Yagasaki, and K. Holmström. LAKE - A Program System for Equilibrium Analytical Treatment of Multimethod Data, Especially Combined Potentiometric and NMR Data. *Acta Chem.Scand.*, 50:717–734, 1996.

[30] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, October 1993.

[31] Donald R. Jones. DIRECT. *Encyclopedia of Optimization*, 2001. To be published.

[32] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive Black-Box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[33] P. Lindström. *Algorithms for Nonlinear Least Squares - Particularly Problems with Constraints.* PhD thesis, Inst. of Information Processing, University of Umeå, Sweden, 1983.

[34] David G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, Massachusetts, 2nd edition, 1984.

[35] J. J. More, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Software*, 7:17–41, 1981.

[36] G. L. Nemhauser and L. A. Wolsey. Integer programming. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science.* Elsevier/North Holland, Amsterdam, The Netherlands, 1989.

[37] J. Petersson. *Algorithms for Fitting Two Classes of Exponential Sums to Empirical Data.* Licentiate Thesis, ISSN 1400-5468, Opuscula ISRN HEV-BIB-OP–35–SE, Division of Optimization and Systems Theory, Royal Institute of Technology, Stockholm, Mälardalen University, Sweden, December 4, 1998.

[38] Axel Ruhe and Per-Åke Wedin. Algorithms for Separable Nonlinear Least Squares Problems. *SIAM Review*, 22(3):318–337, 1980.

[39] K. Schittkowski. On the Convergence of a Sequential Quadratic Programming Method with an Augmented Lagrangian Line Search Function. Technical report, Systems Optimization laboratory, Stanford University, Stanford, CA, 1982.

A Nonlinear Programming Example I - f,g,H

```
% ---------------------------------------------------------------------
% function f = con1_f(x, Prob)
% ---------------------------------------------------------------------
%
% con1_f computes the objective function f(x)
%

function f = con1_f(x, Prob)

% Use the standard TOMLAB global variable to be used for communication
% between the function, gradient and Hessian routines

global US_A

% The value could either be directly set as US_A = exp(x(1));
% or as structure field:

US_A.e1 = exp(x(1));

% If many items are to be globally sent, structure fields are easier to use.

f = US_A.e1 * (4*x(1)^2 + 2*x(2)^2 + 4*x(1)*x(2) + 2*x(2) + 1);

% ---------------------------------------------------------------------
% function g = con1_g(x, Prob)
% ---------------------------------------------------------------------
%
```

```
% con1_g evaluates the gradient g(x)

function g = con1_g(x, Prob)

global US_A

g = US_A.e1 * [4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+8*x(1)+6*x(2)+1;4*x(1)+4*x(2)+2];

% ---------------------------------------------------------------------
% function H = con1_H(x, Prob)
% ---------------------------------------------------------------------
%
% con1_H evaluates the gradient H(x)

function H = con1_H(x, Prob)

global US_A

a = 4*x(1)^2 + 2*x(2)^2 + 4*x(1)*x(2) + 16*x(1) + 10*x(2) + 9;
b = 4*x(2) + 4*x(1) + 6;

H = US_A.e1 * [a b;b 4];
```

B Nonlinear Programming Example I - c,dc

```
% ---------------------------------------------------------------------
% function c = con1_c(x, Prob)
% ---------------------------------------------------------------------
%
% con1_c evaluates the constraints c(x)
%

function c = con1_c(x, Prob)

% Two nonlinear (quadratic) constraints

c = [ - x(1)*x(2) + x(1) + x(2); x(1)*x(2)];


% ---------------------------------------------------------------------
% function dc = con1_dc(x, Prob)
% ---------------------------------------------------------------------
%
% con1_dc evaluates the constraint gradients dc(x)
%

function dc = con1_dc(x, Prob)

% The derivative of the two quadratic constraints are linear functions

% One row for each constraint, one column for each variable

dc = [-x(2)+1 x(2); -x(1)+1 x(1)]';
```