The TOMLAB v2.0 Optimization Environment [1]

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

January 26, 2000

Abstract

TOMLAB is a general purpose, open and integrated Matlab environment for the solution of a wide range of optimization problems, as well as for research and teaching in optimization. One motivation for TOMLAB is to simplify research on applied optimization problems, giving easy access to all types of solvers; at the same time having full access to the power of Matlab. This paper discusses the design and contents of TOMLAB, its performance and presents some applications where it has been successfully used.

More than 65 different algorithms for linear, discrete and nonlinear optimization are implemented. It is also possible to call solvers in the Math Works Optimization Toolbox and a collection of general-purpose Fortran and C solvers using predefined MEX-file interfaces. There are several ways to solve optimization problems in TOMLAB. Either by a direct call to a solver or using a general multi-solver driver routine, or interactively, using the Graphical User Interface (GUI) or a menu system. TOMLAB v2.0 is also call-compatible with the Math Works Optimization Toolbox 2.0.

A large set of standard test problems are included, as well as example and demonstration files. New user-defined problems are easily added. Furthermore, using MEX-file interfaces, problems in the CUTE test problem data base and problems defined in the AMPL modeling language can be solved. TOMLAB v1.0 solves small and medium size dense problems and is free for academic purposes. TOMLAB v2.0 also solves sparse problems, is possible to compile and has enhanced functionality. TOMLAB is running on Unix, Linux and PC systems.

1 Introduction

Many scientists and engineers are using MATLAB as a modeling and analysis tool, but for the solution of optimization problems, the support is weak. That was the motivation for starting the development of TOMLAB, first presented in Holmström [24, 23]. TOMLAB is a general-purpose, open and integrated environment in Matlab for the solution of a wide range of optimization problems, as well as a usable tool for research and teaching in optimization. This paper discusses the design and contents of TOMLAB, its performance and presents some applications where it has been successfully used.

One new idea in TOMLAB is provide the user with several types of interfaces, so the user can choose one that is suitable for the task. The user can use a graphical user interface (GUI), one of a set of of problem specific menu systems, one of a set of multi-solver driver routines, or the user can directly run one of the many solvers. TOMLAB has an integrated approach simplifying for the user to run many different solvers and problems, and giving the user access to a large set of internal TOMLAB solvers. Furthermore it is equally easy for the user to run, using the TOMLAB definition language, other Fortran/C/Matlab solvers. TOMLAB solves all interface problems that otherwise the user would face if trying to run different solvers. It is equally easy to run a more general type of solver for a specific problem, e.g. running a general sparse nonlinear solver on a dense linear or quadratic problem.

[1] Information about TOMLAB available at the Applied Optimization and Modeling **TOM** home page URL: **http://www.ima.mdh.se/tom**.

This paper is organized as follows. An overview of TOMLAB is given in Section 2. Section 3 discusses how TOMLAB is built on top of Matlab. In Section 4 different user objectives are discussed and the design of TOMLAB. The connection with the world outside Matlab using MEX-file interfaces and the current interfaces are the topic of Section 5. The graphical user interface (GUI) and the menu systems are shortly discussed in 6. Section 7 is rather long, presenting linear, discrete and quadratic programming in TOMLAB, but also showing how to run some simple linear programming examples from the command line. This shows how to the design influences the usage of TOMLAB. Section 8 - 10 discusses the solution of nonlinear programming, nonlinear parameter estimation and global optimization. The special treatment and applications of exponential sum fitting problems are presented in Section 11. More applications are presented in 12. Finally, we end with some conclusions in Section 13.

2 Overview of The Optimization Environment TOMLAB

TOMLAB is a Matlab 5.x based optimization environment running on both Unix, Linux and PC systems. One motivation for TOMLAB is to simplify research on practical optimization problems, giving easy access to all types of solvers; at the same time having full access to the power of Matlab. The design principle is: *define your problem once, optimize using any suitable solver.* This is possible using general gateway and interface routines, global variables, string evaluations and structure arrays. When solving advanced applied problems, the design principle is of special importance, as it might be a huge task to rewrite the function, constraint and derivative specifications. The aim of TOMLAB is to provide access to most state-of-the-art numerical optimization software in an integrated and easy-to-use way. TOMLAB could be used as a stand-alone tool or as part of a more general algorithm. For the optimization algorithm developer and the applied researcher in need of optimization tools it is very easy to compare different solvers or do test runs on thousands of problems. A tool is provided to automate the tedious work of making computational result tables from test runs, directly making LaTeX tables for inclusion in papers. TOMLAB should be seen as a proposal for a standard for optimization in Matlab.

In TOMLAB itself about 65 numerical optimization algorithms are implemented, and using different types of predefined interfaces, many more solvers are directly possible to use, see Section 5. Most areas of optimization are covered. There are internal TOMLAB solvers for linear programming, mixed-integer programming, quadratic programming, unconstrained optimization, convex and non-convex nonlinear programming, constrained linear and nonlinear least squares, box-bounded global optimization and global mixed-integer nonlinear programming. The performance of the internal TOMLAB solvers are discussed in more detail in Section 7-10.

A number of routines for special problems are implemented, e.g. partially separable functions, separable nonlinear least squares, dual linear programming, approximation of parameters in exponential models, transportation simplex programming, network simplex programming and binary mixed-integer programming.

A stack strategy for the global variables makes recursive calls possible, in e.g. separable nonlinear least squares algorithms. One structure holds all information about the problem and one holds the results. As discussed in the previous section, there are several ways to solve optimization problems in TOMLAB. Yet another way to solve an optimization problem in TOMLAB v2.0 is to use the call-compatible interfaces simulating the behaviour of the Math Works Optimization Toolbox 2.0, which is especially useful for users that previously used that toolbox. The GUI may also be used as a preprocessor to generate Matlab code for stand-alone runs.

If analytical derivatives are not available, automatic differentiation is easy using an interface to ADMAT/ADMIT TB, free for academic use, see the URL: **http://simon.cs.cornell.edu/home/verma/AD/**. Furthermore, five methods for numerical differentiation are implemented. The default numerical differentiation type is an implementation of the FD algorithm [18, page 343]. It is the classical approach with forward or backward differences, together with an automatic step selection procedure. If the Spline Toolbox is installed, gradient, Jacobian, constraint gradient and Hessian approximations could be computed in three different ways using either of the routines *csapi*, *csaps* or *spaps*. Practical experience shows that this approach works well. Numerical differentiation is automatically used for gradient, Jacobian, constraint gradient and Hessian if the user routine is not defined, but is also selectable. Many standard test problems are included, as well as example and demonstration files. New user-defined problems are easily added.

Included in TOMLAB are interfaces to a number of the solvers in the Math Works Optimization Toolbox 1.x [9] and 2.0 [10], see the TOMLAB User's Guide [27, 31]. Depending on how the paths are set, either the TOMLAB call compatible solvers or the Math Works Optimization TB v2.0 solvers are called. Using MEX-file interfaces, problems in the CUTE test problem data base and problems defined in the AMPL [16] modeling language can be solved, see Section 5.

TOMLAB v1.0 handles small and medium size dense problems, is free for academic use and is possible to download at the home page of the Applied Optimization and Modeling (TOM) group, URL: **http://www.ima.mdh.se/tom**. No support is given. TOMLAB v1.0 is based on NLPLIB TB, a Matlab toolbox for nonlinear programming and parameter estimation [29], and OPERA TB, a Matlab toolbox for linear and discrete optimization [30]. A User's Guide for TOMLAB v1.0 is available [31]. The size of TOMLAB v1.0 is quite large; only the Matlab code consists of more than 300 m-files and more than 55 000 lines of code.

The new TOMLAB v2.0 system also handles large, sparse problems and the code is possible to automatically convert to C++ and compile and run faster than in Matlab using the MIDEVA system. Both TOMLAB v2.0 and MIDEVA are available from the web site of MathTools Inc, URL: **http://www.mathtools.com**. A User's Guide for TOMLAB v2.0 [27], is downloadable at the URL: **http://www.ima.mdh.se/tom**. The toolbox consists of more than 80 000 lines of Matlab code in about 420 files.

3 TOMLAB and Matlab

The TOMLAB optimization environment aims at the solution of a broad range of different optimization problems. Each type of problem demands a lot of problem dependent information to be stored and distributed throughout the system. The TOMLAB solution is to use structure arrays for all information storage. One array, *Prob*, stores all information about the problem to be solved, and one array, *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, described in detail in [26] and the User's Guides [31, 27], In the TOMLAB system only the pointers to these structures are necessary to pass around. All user defined routines (for nonlinear problems) have access to the full structure, because this structure variable is an argument to all low level routines. If the user needs to supply information to the low level routines, the recommended solution is to set this information as arbitrarily sub fields to *Prob.USER*. By this way, information needed to evaluate the low level routines is easily retrieved. All TOMLAB solvers are written to use the structure arrays. Each external solver needs an interface routine that unpacks the structure array *Prob*, makes the call to the solver, and then creates the resulting structure *Result*. That means that 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. The use of structure arrays make advanced result presentation and statistics possible. In Section 7 some examples of how to run TOMLAB to solve linear programming problems illustrates the design.

4 The Design of TOMLAB

Before going more into the design of TOMLAB, it is of interest to identify the objective of the optimization task that the potential TOMLAB user might have. We have identified some general objectives:

1. Solve a single well-defined optimization problem, as part of some industrial or academic project. The user most often is using a simple command line call to a particular solver.

2. Solve a large set of a certain class of optimization problems, as part of some industrial or academic project. The GUI is useful for working on test problems and finding the best choice of optimization parameters and solvers. Later the user wants to use the best available solver from the command line going through the whole set of problems.

3. Develop new algorithms for any of the problem areas covered. It is then interesting to compare the results with other solvers and run large tests from the command line, maybe also running problems from the CUTE test set and problems defined in the AMPL language.

4. Develop new algorithms for some problem class not covered by TOMLAB, which needs sub-solutions covered by the TOMLAB solvers. The user wants to run command line calls. To find the most suitable solver, the multi-driver interface is best to use, to very easily change sub-solver.

5. Demonstrate optimization methods and algorithms for students in optimization. Show algorithmic behaviour on different kinds of problems. The GUI and the menu systems gives the student fast access to the full system.

The graphical utilities showing contour plots with search steps and line search attempts gives the student a good feel for how algorithms work.

6. Solve time-critical optimization problems, for example as part of a control algorithm. The user wants to eliminate all overhead and directly call the fastest and most robust available solver. To get fastest speed, most often a MEX-file call is necessary. The two-layer approach in the TOMLAB MEX-file interfaces makes a direct call possible.

7. Develop new algorithms for demanding applied optimization problems, where the specific selection of sub-solvers and optimization parameters are not clear-cut. Different solvers have different properties, and may be better or worse suitable for the problem in mind. Numerical tests on the actual problem are needed to find the definite solution. Both GUI and the multi-solver driver routines are good tools to use.

8. Solve nonlinear parameter estimation problems. The graphical utilities available in the GUI and menu systems makes it possible to see how the model fits to data. The problems are often ill-conditioned and badly scaled, and verifying that the true optimum is found is often necessary. The possibility to directly rerun the optimization from the solution point using any solver is useful in the verifying process.

4.1 The Different Types of Optimization Problems

TOMLAB solves a number of different types of optimization problems. Currently, the types listed in Table 1 are defined. Both a string identifier, up to three characters long, and a number, is used to identify the problem class. As the system is further developed, and new types of solvers are added, this list is to be increased. The TOMLAB variable *probType*, one field in the structure *Prob*, defines the the current problem type to be solved. Beside these predefined problem classes, TOMLAB includes a selection of solvers for special linear and discrete problems, described in Section 7.3.

An optimization solver is defined to be of type *solvType*, where *solvType* is any of the *probType* entries in Table 1. It is clear that a solver of a certain *solvType* is able to solve a problem defined to be of another type, if the problem class is a subclass of the larger one. However, the general solver might be more or less suitable for the subclass. Table 2 shows the different possibilities.

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	21	Exponential fitting problems.
nts	22	Nonlinear time series.

4.2 Solver Routines in TOMLAB

In Table 3 the internal optimization solvers in TOMLAB are listed. External solvers are discussed in Section 5. The exact problem formulations handled by the solvers are defined in Section 7-10. The design is made easy expandable for more problem types, more solver types and more solvers. Special interface routines are written for the the following four sub-problems: linear programs, Phase 1 feasible point determination, dual linear programs and quadratic programs. All internal solvers that as part of the algorithm solve sub-problems of any of these four types, are calling the interface routine. That makes it possible to switch the sub-solver used by changing a field in the input problem structure. Thus any internal solver may use external solvers for the four sub-problems, e.g. the

Table 2: The problem classes (*probType*) possible to solve with each type of solver (*solvType*) is marked with an x. When the solver is in theory possible to use, but from a practical point of view is probably not suitable, parenthesis are added (x).

	SolvType									
probType	**uc**	**qp**	**con**	**ls**	**lls**	**cls**	**mip**	**lp**	**glb**	**glc**
uc	x		x						x	(x)
qp		x	x							(x)
con			x							(x)
ls			x	x		x				(x)
lls		x	x	x	x	x				(x)
cls			x			x				(x)
mip							x			(x)
lp		x	x				x	x		(x)
glb			(x)						x	x
glc			(x)							x
exp	x		x	(x)		x				(x)

constrained nonlinear solvers may use a Fortran solver like QPOPT (Section 5) to solve quadratic programs fast and robust. To get a list of all available solvers for a certain problem type, including Fortran, C and the Math Works Optimization Toolbox solvers, a routine *PrintSolvers* is available. The line search algorithm *LineSearch* used by the solvers *conSolve*, *lsSolve*, *clsSolve* and *ucSolve* is a modified version of the algorithm by Fletcher [14, chap. 2]. Bound constraints are treated as described in Gill et. al. [18].

TOMLAB includes a large set of predefined test problems for each problem class. They are usable for a user developing algorithms, as well as for system testing and demonstration. Define *probSet* to be a set of predefined optimization problems to be solved. Each *probSet* belongs to a certain class of optimization problems of type *probType*. For each *probSet* one *Problem definition file* is defined that initializes each problem. TOMLAB v2.0 defines a data base with all *Problem definition files*, that is used by the menu systems and the GUI. New users want to add their own problems collected into new Problem definition files. TOMLAB v2.0 includes operations to add problem definition files (*AddProblemFile*) and delete problem definition files (*DeleteProblemFile*) in the data base. To create new Problem definition files, there are TOMLAB v2.0 routines that automatically make the Matlab code for a new set of test problems. In TOMLAB v1.0 the user must edit the data base and Problem definition files directly, as described in the User's Guide.

The function in the *Problem definition file* has two modes of operation; either return a string matrix with the names of the problems in the *probSet* or the *Prob* structure with all information about the selected problem. Using a structure makes it easy to add new items without too many changes in the rest of the system. The menu systems and the GUI are using the string matrix for user selection of which problem to solve.

4.3 Low Level Routines and Test Problems

In the low level routines the mathematical descriptions of the optimization problems are implemented. These routines compute the objective function value, the residual vector (for nonlinear least squares (NLLS) problems), the vector of constraint functions, and the first and second order derivatives of these functions. The generic names recommended are shown in Table 4. Only the routines relevant for a certain type of optimization problem need to be coded. There are dummy routines for the other routines. If routines that are computing derivatives are undefined (function name variables are set as empty), TOMLAB automatically uses numerical differentiation.

Different solvers all have very different demand on how the sufficient information should be supplied, i.e. the function to optimize, the gradient vector, the Hessian matrix. To be able to code the problem only once and then use this formulation to run all types of solvers, interface routines has been developed that returns the information in the format needed for the actual solver.

Table 3: Internal 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 [37].
sTrustr	con	Convex optimization of partially separable functions, using a structural trust region algorithm [11], combined with an initial trust region radius algorithm [43].
glbSolve	glb	The DIRECT algorithm [34]. Stand-alone version *glbSolve*.
glcSolve	glc	Extended DIRECT [35]. Stand-alone version *glcSolve*.
ego	glb	The Efficient Global Optimization (EGO) algorithm [36].
llsSolve	lls	Linear least squares using an active-set strategy.
clsSolve	cls	Active-set strategy [6]. Implements the Gauss-Newton method, the Al-Baali-Fletcher [3] and the Fletcher-Xu [13] hybrid method, and the Huschens TSSM method [32]. Subspace minimization handles rank problems.
conSolve	con	Sequential quadratic programming (SQP) method by Schittkowski [44]. Han-Powell SQP method.
nlpSolve	con	Filter SQP algorithm by Fletcher and Leyffer [15].
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 [40, chap. 8].
cutplane	mip	Cutting plane algorithm using Gomorov cuts.

4.4 The Functional Structure of TOMLAB

A flow-sheet of the process of optimization in TOMLAB is shown in Figure 1. Normally, a single optimization problem is solved running any of the menu systems (one for each *solvType*), or using the Graphical user interface (GUI). When several problems are to be solved, e.g. in algorithmic development, it is inefficient to use an interactive system. This is symbolized with the *Advanced user* box in the figure, which directly runs the *Optimization driver*. The user may also directly call any TOMLAB solver. 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 does the book-keeping, keep track of search directions, and determines type of differentiation; analytic, automatic, or any of the different types of numerical differentiation.

A set of Matlab m-files are needed to implement the chain of function calls for all solver types and problem sets, i.e. for the menu systems, driver routines etc. Table 5 shows the naming convention. The names of the *Problem definition file* and the low level routines are constructed as two parts. The first part being the abbreviation of the relevant *probSet*, and the second part denotes the computed task, shown in Table 4.

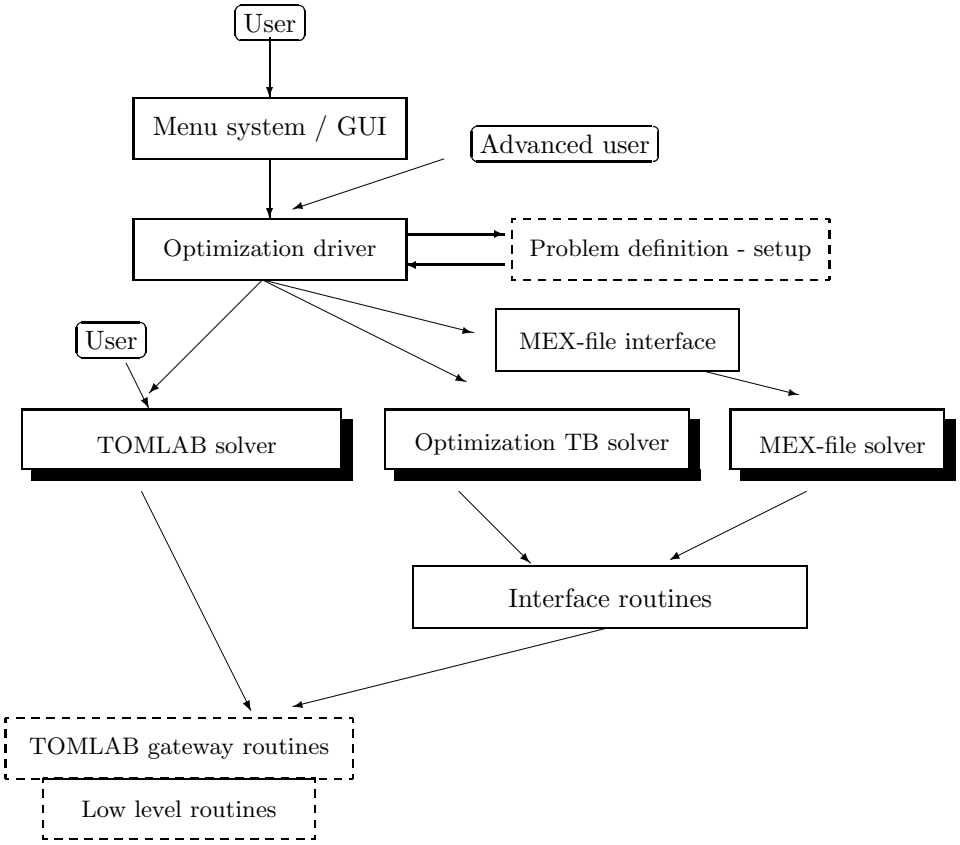

Figure 1: The process of optimization in TOMLAB.

Table 4: Names on the low level m-files in **NLPLIB TB**.

Generic name	**Purpose** ($\diamond$ is any *probSet*, e.g. $\diamond$=**con**)
$\diamond$_prob	Problem definition file. Define string matrix with problems or a structure *Prob* for the selected problem.
$\diamond$_f	Compute objective function $f(x)$.
$\diamond$_g	Compute the gradient vector $g(x)$.
$\diamond$_H	Compute the Hessian matrix $H(x)$.
$\diamond$_c	Compute the vector of constraint functions $c(x)$.
$\diamond$_dc	Compute the matrix of constraint normals, $\partial c(x)/dx$.
$\diamond$_d2c	Compute the 2^{nd} part of 2^{nd} derivative matrix of the Lagrangian function, $\sum_i \lambda_i \partial^2 c_i(x)/dx^2$.
$\diamond$_r	Compute the residual vector $r(x)$.
$\diamond$_J	Compute the Jacobian matrix $J(x)$.
$\diamond$_d2r	Compute the 2^{nd} part of the Hessian matrix, $\sum_i r_i(x) \partial^2 r_i(x)/dx^2$

Table 5: Names of main m-file functions in TOMLAB.

Generic name	**Purpose** (*solvType* is $\diamond$, e.g. $\diamond$=**con**)
$\diamond$Opt	Menu program.
$\diamond$Run	Multi-solver optimization driver routine.
$\diamond$Def	Routine defining default optimization parameters.
$\diamond$Solve	Solver.
$\diamond$Assign	Define the *Prob* structure for an optimization problem, also the problem definition file.
$\diamond$VarDef	Utility defining problem relevant parameters
$\diamond$ProbSet	Utility setting variables in Prob structure

5 The MEX-file Interfaces

The last forty years an intensive research in the area of optimization has lead to the development of many high-quality solvers, most often written in Fortran. Using the MEX-file interface facility in Matlab it is possible to develop interfaces to programs written in C, C++ and Fortran. From the users point of view, no difference is seen, except for increased speed in the solution process. If an error occurs the error message is difficult to interpretate, but for a well-tested system that seldom occurs. There is however a trade-off between extensive user input testing and speed of solution process. Normally a user wants high-speed performance when running a solver using the MEX-file interface, and making a lot of testing in Matlab often takes as much time as the actual solution process.

In TOMLAB v2.0 the MEX-file interfaces have been developed in two layers. The upper level unpacks the problem definition structure, makes some input checking and calls the lower level. The lower level has minimal input checking and just calls the MEX-file interface routine. That makes it possible to directly use the lower level call when speed is critical, a case for the User Type 6 mentioned in Section 4. Presently, MEX-file interfaces have been developed for the solvers in Table 6. The MEX-file interfaces have been tested on Unix, Linux and PC systems.

5.1 The AMPL Interface

The modeling language AMPL [16] is using ASCII files to define a problem. These files are normally converted to binary files with the extension *nl*, called *nl*-files. The current TOMLAB AMPL interface is an interface to problems defined in the AMPL *nl*-format and is using the rudimentary Matlab interface written in C by Gay [17]. When using the TOMLAB AMPL interface, the user either gets a menu of the *nl*-files found or directly makes a choice of which problem to solve.

Table 6: Current MEX-file interfaces in TOMLAB.
An S in the S/D column means a sparse solver, and D a dense solver.

Solver Name	Type	S/D	Developers
MINOS 5.5	**con**	S	B. Murtagh and M.A. Saunders.
MINOS 5.6	**con**	S	B. Murtagh and M.A. Saunders.
SNOPT 5.3-4	**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 5.3-4	**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.
XPRESS/MP	**mip**	S	Dash Associates Ltd.
BTNCLC 1.0	**con**	D	H Schramm. (Nonsmooth $f(x)$, linear constraints)

5.2 The CUTE Interface

The Constrained and Unconstrained Testing Environment (CUTE) [7, 8] is a well-known software environment for nonlinear programming. The distribution of CUTE includes a test problem data base of nearly 1000 optimization problems, both academic and real-life applications. This data base is often used as a benchmark test in the development of general optimization software. Using the TOMLAB CUTE interface it is possible to run the huge set of CUTE test problems using any solver. The TOMLAB interface is based on the Matlab interface that is part of the CUTE distribution. When using the TOMLAB CUTE interface, the user either gets a menu of all precompiled files in the CUTE directory chosen, or directly makes a choice of which problem to solve.

6 The Graphical User Interface and Menu Systems

Running the GUI or the different menu systems, the user can select the problem to be solved and the solver to be used and change the default values for most optimization and solver parameters. The GUI may also be used as a preprocessor to generate Matlab code for command line runs [12].

There are options to display graphical information for the selected problem. For two-dimensional problems, graphical display of the optimization problem as mesh or contour plot is possible. In the contour plot, the iteration steps, trial step lengths, and constraints are displayed. For higher-dimensional problems, iteration steps are displayed in two-dimensional subspaces. Special plots for nonlinear least squares problems are available, e.g. plot of model against data.

7 Linear, Quadratic and Discrete Optimization

The formulation in TOMLAB for a linear program (**lp**) 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 $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b_L, b_U \in \mathbb{R}^m$. Equality constraints are defined by setting the lower bound equal to the upper bound. Linear programs are solved using the TOMLAB routine *lpSolve*. It is also possible to

use the active-set quadratic programming method in *qpSolve* or a general nonlinear solver like *nlpSolve* or *conSolve*. Using a MEX-file interface, the SOL routines *MINOS* or *QPOPT* are suitable for linear programming problems. To choose alternative solvers, the multi-driver routine *lpRun* is called. In the following the use of TOMLAB from the command line is illustrated for linear programming problems. The approach is similar when solving other types of problems. Consider the solution of a very simple linear programming test problem

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

named *LPTest*. The following statements define this problem in Matlab

```
Name  = 'LPTest';
c     = [-7 -5]';
A     = [ 1  2
          4  1 ];
b_U   = [ 6 12 ]';  b_L   = [-inf -inf]';
x_L   = [ 0  0 ]';  x_U   = [];              x_0   = [];
```

If the above statements are defined in a file *lpExample.m*, the simplest way to solve this problem in TOMLAB 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);
% Run QP solver
Result1 = qpSolve(Prob);
% Run NLP solver
Result2 = nlpSolve(Prob);
```

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 is used to define the standard Prob structure, which TOMLAB always uses to store all information about a problem. One advantage is that it is very easy to call other solvers, once the problem is defined in the general TOMLAB structure format. This is shown by the last statements in the example, calling a quadratic (*qpSolve*) and a constrained nonlinear solver (*nlpSolve*). In this test the initial value x_0 for the decision parameters is empty, so an LP Phase I problem is solved to find a feasible point.

TOMLAB is also call compatible with Math Works Optimization TB v2.0. This means that you can use exactly the same syntax, but the solver is a TOMLAB solver instead. For LP solver *linprog* in the TOMLAB version adds one extra input argument, the *Prob* structure, and one extra output argument, the *Result* structure. Both extra parameters are optional, but implies that the additional functionality of the TOMLAB LP solver is accessible. An example of the use of the TOMLAB *linprog* solver to solve test problem (2) illustrates the difference in the two approaches. *linprog* needs linear inequalities and equalities to be given separately. The two sets are easily identified.

```
lpExample;
ix = b_L==b_U;
E  = find(ix);
I  = find(~ix);
[x, fVal, ExitFlag, Out, Lambda] = linprog(c, A(I,:), b_U(I),...
    A(E,:), b_U(E), x_L, x_U, x_0);
```

If the user wants to solve more than one LP, or maybe the same LP but under different conditions, then it is possible to define the problems in the TOMLAB *Problem definition file* format. This gives full access to all different solvers, menus, driver routines and the GUI. Using the same example (2) to illustrate how this is done in TOMLAB gives the Matlab statements

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

Two more parameters are needed in the call to *lpAssign*. The first tells *lpAssign* to define a new TOMLAB *Problem definition file* called *lptest*. The second, now set as one, is the number of problems to be defined.

The *lptest* problem file is included in the TOMLAB *Problem definition file* data base as a predefined test file by the call to *AddProblemFile* and is automatically set as the default file for LP problems. Calling *lpRun* with only the name of the solver runs the first test problem in the default LP file, i.e. the first problem in file *lptest*, which is the problem just defined. Thus, the result will be exactly the same as for the first example.

In the call to *lpAssign* the parameter after the file name *lptest*, called *nProblem*, controls which of two types of definition files are created. Setting this parameter as empty or one, as in this case, defines a program structure in the file *lptest* in which the user easily can insert more test problems. The 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 to define the problem in the file **lptest.m**.

When doing this automatic *Problem definition file* generation *lpAssign* stores the problem in the Matlab binary mat-file format with a name combined by three items: the name of the problem file (*lptest*), the string '_P' and the problem number given as the input parameter next after the problem file name. In this case *lpAssign* defines the file *lpprob_P1.mat*.

It is easy to create a *Problem definition file* for a large set of test problems. This feature is illustrated by running a test where the test problem is disturbed by random noise. The vector c in the objective function are disturbed, and the new problems are defined and stored in the TOMLAB *Problem definition file* format. To simplify restrict the large number of test problems to be three.

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

Each problem gets a unique name. In the first iteration, $i = 1$, *lpAssign* defines a *Problem definition file* with three test problems, and defines the first test problem, stored in *lplarge_P1.mat*. In the other iterations *lpAssign* just defines the other mat-files, *lplarge_P2.mat* and *lplarge_P3.mat*.

AddProblemFile adds the new *lplarge* problem as the default LP test problem in TOMLAB. The *runtest* test program utility runs the selected problems, in this case all three defined. The second zero argument is used if the actual solver has several algorithmic options. In this case the zero refers to the default option in *lpSolve*, to use the minimum cost rule as the variable selection rule. The last arguments are used to lower the default output and avoid a pause statement after each problem is solved. The results are shown in the following file listing.

```
Solver: lpSolve. Algorithm 0
```

```
=== * * * ================================================ * * *
Problem  1: LPTest - 1              f_k     -26.501771581492278000
=== * * * ================================================ * * *
Problem  2: LPTest - 2              f_k     -26.546357769596234000
=== * * * ================================================ * * *
Problem  3: LPTest - 3              f_k     -26.425877951614158000
```

We have made tests to compare the efficiency of different solvers on dense medium size LP problems. The solver *lpSolve*, two algorithms implemented in the solver *linprog* from Optimization Toolbox 2.0 [10] and the Fortran solvers MINOS and QPOPT are compared. MINOS and QPOPT are called using the MEX-file interface. In all test cases the solvers converge to the same solution. The results of similar tests for TOMLAB v1.0 are presented in more detail in [30]. Here we present a summary of the results in Table 7, the mean values for each of the five groups of tests. All tables are found in the User's Guide [27].

When comparing elapsed computational time it is clear that *lpSolve* is remarkable faster then the corresponding simplex algorithm implemented in the *linprog* solver, about a factor eleven to sixty-three times faster. *lpSolve* is also a factor 2.3 to twenty-two times faster than the other algorithm implemented in *linprog*, a primal-dual interior-point method aimed for large-scale problems. It is clear that if speed is critical, the availability of Fortran solvers callable from Matlab using the MEX-file interfaces in TOMLAB are very important.

Table 7: Computational results on randomly generated medium size LP problems for four different routines. $Iter$ is the number of iterations and $Time$ is the elapsed time in seconds on a Dell Latitude CPi 266XT running Matlab 5.3. The *lpS* solver is the TOMLAB v2.0 *lpSolve* with the minimum cost selection rule (iterations It_m, time T_m). The time T_c is the execution time with *lpSolve* compiled and ran in MIDEVA. The *linprog* solver in the Math Works Optimization Toolbox 2.0 implements two different algorithms, a medium-scale simplex algorithm (time T_m) and a large-scale primal-dual interior-point method (iterations It_l, time T_l). MINOS and QPOPT are called using the TOMLAB MEX-file interface. The number of variables, n, the number of inequality constraints, m, the objective function coefficients, the linear matrix and the right hand side are chosen randomly. Each row presents the mean of a test of 20 test problems with mean sizes shown in the first two columns.

n	m	*lpS*	*linprog*	*lpS*	*lpS*	*Minos*	*qpopt*	*linprog*	*linprog*
		It_m	It_l	T_c	T_m	$Time$	$Time$	T_m	T_l
103	45	8	13	0.22	0.52	0.30	0.30	5.70	1.23
206	149	11	20	1.75	2.01	0.50	0.39	32.18	12.91
308	200	11	23	3.15	3.36	0.75	0.50	98.18	47.20
402	243	10	24	4.15	4.46	0.98	0.63	204.99	94.11
503	243	14	29	5.89	6.05	1.26	0.94	383.16	132.28

7.1 Quadratic Programming Problems

The formulation in TOMLAB for a quadratic program (**qp**) is

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x^T F x + c^T x \\ s/t \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array} \end{aligned} \tag{3}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m\times n}$, and $b_L, b_U \in \mathbb{R}^m$. Indefinite quadratic programs are solved with the active-set method implemented in *qpSolve*, which is also the default QP solver for the nonlinear TOMLAB solvers *conSolve* and *nlpSolve*. It is also possible to choose a general constrained solver or the SOL routine *QPOPT* using a MEX-file interface. To choose alternative solvers, the multi-driver routine *qpRun* is called.

7.2 Mixed-Integer Programming

The formulation in TOMLAB for a mixed-integer program (**mip**) is

$$\begin{array}{lll} \min\limits_{x} & f(x) = c^T x & \\ s/t & \begin{array}{rcl} x_L & \leq & x \leq x_U, \\ b_L & \leq & Ax \leq b_U,\ x_j \in \mathbb{N} \quad \forall j \in I \end{array} \end{array} \tag{4}$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b_L, b_U \in \mathbb{R}^m$. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers. Mixed-integer programs are normally solved using the TOMLAB routine *mipSolve*. When dual feasible solutions are available, *mipSolve* is using the TOMLAB dual simplex solver *DualSolve* to solve sub problems, which is significantly faster than using an ordinary linear programming solver, like the TOMLAB *lpSolve*. Another alternative solver is the cutting plane routine *cutplane*. Available is also a MEX-file interface to XPRESS/MP. To choose alternative solvers, the multi-driver routine *mipRun* is called.

User defined priorities for variable selection, as well as different tree search strategies, are implemented in *mipSolve*. For 0/1 - knapsack problems a round-down primal heuristic is included. The test results on nodes visited (iterations) in Table 8 shows how much the result is influenced by using priorities and the round-down heuristic on three test problems for 0/1 - knapsack problems (Weingarter 1, Hansen Plateau 1, PB 4). The priority vector is set to the objective function coefficients (lowest value gives highest priority). The last two columns show results for the XPRESS/MP solver, when using and not using an aggressive branch-and-cut strategy, which gives more work at each node, but less nodes visited.

Table 8: Tests on knapsack problems. The numbers are nodes visited (iterations). XP is the XPRESS/MP Release 11 solver called using the TOMLAB v2.0 MEX-file interface.

Knapsacks/ Variables/	mipSolve	mipSolve Heuristic	mipSolve Priorities	mipSolve Heuristic Priorities	XP Use Cuts	XP No Cuts
2/28	896	140	470	94	1	135
4/28	1046	752	130	76	103	29
2/29	2988	822	1228	208	48	292

7.3 Other Tools for Linear and Discrete Optimization (LDO)

TOMLAB v2.0 also includes a collection of additional tools for linear, quadratic and discrete optimization. These tools are stand-alone tools and are not integrated with the GUI, menu systems and multi-driver routines. Included are example files illustrating the use of each of the tools. In Table 9 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 more details, see the TOMLAB User's Guide [27]. In TOMLAB v1.0 these tools were part of the OPERA Toolbox [30]. The solver *cutstock* is new, and *salesman* is rewritten.

8 Nonlinear Optimization

In the standard definition in TOMLAB for unconstrained optimization (**uc**) problems simple bounds are added, important from a practical point of view, i.e.

$$\begin{array}{lll} \min\limits_{x} & f(x) & \\ s/t & x_L \leq x \leq x_U, & \end{array} \tag{5}$$

where $x, x_L, x_U \in \mathbb{R}^n$ and $f(x) \in \mathbb{R}$. The TOMLAB routine *ucSolve* implements several of the most popular search step methods for unconstrained optimization. Another TOMLAB solver suitable for unconstrained problems is the structural trust region algorithm *sTrustr*, which solves convex programs.

Table 9: Some of the linear and discrete (LDO) solvers in TOMLAB v2.0.

Function	Description
karmark	Karmakar's projective algorithm [4, page 386].
akarmark	An affine scaling variant of Karmakar's method [19, pages 411-413]. Purification algorithm [4, page 385].
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 [38, 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 [20].
balas	Branch and bound algorithm for binary IP using Balas method. [22].
dpknap	Forward recursion dynamic programming for knapsack problems.
cutstock	Cutting stock algorithm using revised simplex and *dpknap*.

The constrained nonlinear optimization problem (**con**) is defined as

$$\begin{array}{lllllll} \min\limits_{x} & f(x) & & & & & \\ & & & & & & \\ & x_L & \leq & x & \leq & x_U, \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{6}$$

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 $c_L, c(x), c_U \in \mathbb{R}^{m_2}$. For general non-convex constrained nonlinear optimization the internal TOMLAB solvers *conSolve* and *nlpSolve* are suitable. Several standard solvers are available using MEX-file interfaces. Most solvers handle non-convex problems, but only global convergence to a local minimum is guaranteed. For convex programs *sTrustr* could be used.

9 Global Optimization

There are three different routines for global optimization in TOMLAB, all using only function values. The routine *glbSolve* implements an algorithm for box-bounded global optimization (**glb**), i.e. problem (5) with finite simple bounds on all the variables. *glbSolve* implements the DIRECT algorithm [34], which is a modification of the standard Lipschitzian approach that eliminates the need to specify a Lipschitz constant. For global mixed-integer nonlinear programming (**glc**), a nonconvex problem of type (6) with some x integer constrained, *glcSolve* implements an extended version of DIRECT [35]. When the function is very expensive to compute it is important to choose the next sample function value carefully. The routine *ego* implements the Efficient Global Optimization (EGO) algorithm [36]. 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 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 2. 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 2, 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.

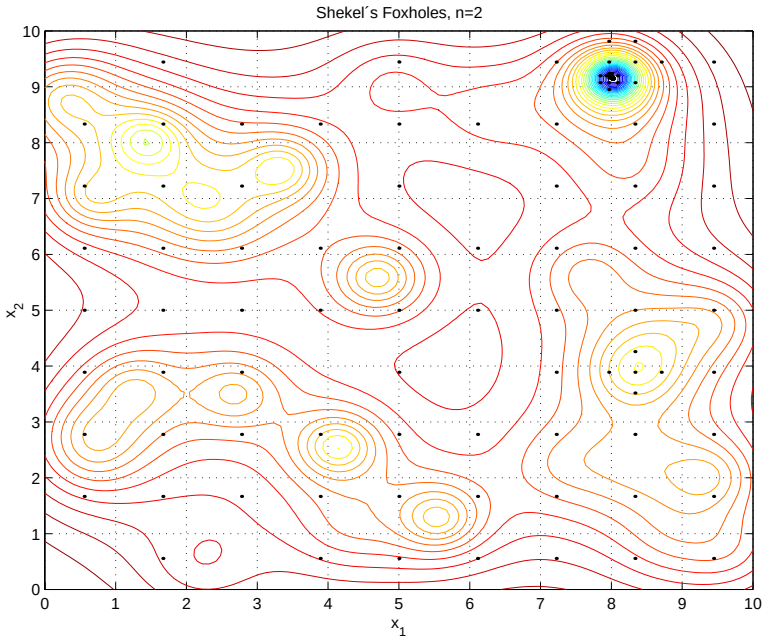

Figure 2: Contour plot of the Shekels's Foxholes function with the 147 sampled points needed for convergence using *glbSolve*.

10 Nonlinear Parameter Estimation and Nonlinear Least Squares

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

$$\begin{array}{lrcl} \min\limits_{x} & f(x) = \frac{1}{2} r(x)^T r(x) & & \\ & & & \\ & x_L \leq & x & \leq x_U, \\ s/t & b_L \leq & Ax & \leq b_U \\ & c_L \leq & c(x) & \leq c_U \end{array} \tag{7}$$

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 11.

A standard test set for nonlinear squares problems is the test set of More, Garbow and Hillstrom [39] with 35 problems. A summary of a comparison with different solvers is shown in in Table 10. 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.

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 12. Once again the TOMLAB *clsSolve* FX is the fastest and most reliable solver. Full test results for both test sets are shown in Table 11 and Table 13.

A comparison of *clsSolve* and other solvers on 26 linearly constrained nonlinear least squares test problems are shown in Table 14. 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 shown in Table 15. For more details on the test problems, and tests on TOMLAB v1.0 solvers, see the thesis of Björkman [6].

Table 10: 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

Table 11: Comparison of algorithmic performance for the More, Garbow, Hillstrom test set for nonlinear least squares problems.

	clsSolve GN	clsSolve FX	lsqnonlin LM	lsqnonlin LS	NLSSOL	NPOPT
MGH 01	18 33 33	19 36 36	26 27 27	24 24 24	32 58 58	24 29 29
MGH 02	15 280 252	8 10 10	134 135 135	31 31 31	9 15 15	7 10 10
MGH 03	15 17 17	15 17 17	82 83 83	—	50015521552 ∗	—
MGH 04	13 42 42	14 32 32	—	—	15 25 25	—
MGH 05	6 8 8	6 8 8	13 14 14	9 9 9	12 15 15	17 22 22
MGH 06	—	14 43 42	13 14 14	26 26 26	23 87 87	25 54 54
MGH 07	8 10 10	8 10 10	19 20 20	—	20 29 29	32 37 37
MGH 08	5 6 6	5 6 6	14 15 15	250 250 250	11 15 15	17 23 23
MGH 09	1 2 2	1 2 2	3 4 4	3 3 3	2 3 3	5 10 10
MGH 10	7 11 11	7 11 11	—	—	204 536 536	—
MGH 11	115 301 301	250 317 317	—	—	388 910 910	—
MGH 12	5 6 6	5 6 6	11 12 12	36 36 36	9 12 12	28 35 35
MGH 13	11 12 12	11 12 12	16 17 17	—	27 28 28	62 69 69
MGH 14	49 93 93	57 77 77	79 80 80	—	47 79 79	40 51 51
MGH 15	15 19 19	7 11 11	16 17 17	128 128 128	13 29 29	21 35 35
MGH 16	50014501450 ∗	17 28 28	39 40 40	141 141 141	454 928 928	18 25 25
MGH 17	8 11 11	11 24 18	30 31 31	—	37 69 69	45 71 71
MGH 18	—	63 119 113	84 85 85	—	—	—
MGH 19	12 17 17	13 20 20	22 23 23	38 38 38	20 33 33	61 88 88
MGH 20	7 8 8	7 8 8	17 18 18	—	18 26 26	39 46 46
MGH 21	18 33 33	18 34 34	32 33 33	21 21 21	32 57 57	63 109 109
MGH 22	11 12 12	11 12 12	18 19 19	12 12 12	28 29 29	147 163 163
MGH 23	62 73 73	19 32 32	70 71 71	14 14 14	143 382 382	146 207 207
MGH 24	144 334 325	155 272 271	—	—	—	338 463 463
MGH 25	10 11 11	10 11 11	6 7 7	11 11 11	15 16 16	18 20 20
MGH 26	8 15 15	8 15 15	15 16 16	—	10 22 22	—
MGH 27	14 46 46	9 30 30	—	12 12 12	13 19 19	18 21 21
MGH 28	2 3 3	2 3 3	5 6 6	—	4 5 5	20 36 36
MGH 29	3 4 4	3 4 4	5 6 6	5 5 5	4 5 5	7 8 8
MGH 30	4 5 5	4 5 5	8 9 9	7 7 7	6 7 7	22 40 40
MGH 31	5 6 6	5 6 6	7 8 8	8 8 8	8 9 9	—
MGH 32	1 2 2	1 2 2	9 10 10	2 2 2	1 2 2	1 2 2
MGH 33	1 2 2	1 2 2	10 11 11	2 2 2	4 9 9	2 4 4
MGH 34	1 2 2	1 2 2	10 11 11	2 2 2	4 9 9	2 4 4
MGH 35	—	104 598 592	—	66 66 66	—	—
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

Table 12: 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

Table 13: Comparison of algorithmic performance for the More, Garbow, Hillstrom test set for nonlinear least squares problems, with additional simple bounds by D. Gay.

	clsSolve GN	clsSolve FX	lsqnonlin LS	NLSSOL	NPOPT
MGH 01	27 39 39	30 44 44	29 29 29	13 18 18	19 24 24
MGH 02	4 5 5	4 5 5	11 11 11	7 9 9	3 4 4
MGH 06	—	5 8 8	13 13 13	9 15 15	13 23 23
MGH 07	10 11 11	8 9 9	73 73 73	10 12 12	11 13 13
MGH 08	5 6 6	5 6 6	—	8 9 9	17 27 27
MGH 10	84 136 115	10 18 13	—	—	—
MGH 12	6 7 7	6 7 7	14 14 14	8 9 9	12 13 13
MGH 13	21 22 22	10 11 11	62 62 62	17 18 18	26 31 31
MGH 15	13 14 14	8 9 9	65 65 65	13 19 19	23 29 29
MGH 16	248 561 561	21 28 28	84 84 84	12 23 23	10 16 16
MGH 17	9 12 12	9 14 14	—	—	118 194 194
MGH 19	30 34 34	39 55 43	—	25 32 32	65 83 83
MGH 20	7 8 8	7 8 8	—	9 10 10	28 34 34
MGH 27	13 15 15	9 11 11	57 57 57	8 10 10	19 24 24
MGH 32	2 3 3	2 3 3	7 7 7	1 2 2	1 2 2
MGH 33	3 4 4	3 4 4	6 6 6	3 6 6	2 4 4
MGH 34	8 9 9	8 9 9	17 17 17	1 2 2	1 2 2
MGH 35	—	—	—	49 124 124	45 66 66
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

Table 14: 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

Table 15: Comparison of algorithmic performance for constrained nonlinear least squares problems.

	clsSolve GN	clsSolve FX	clsSolve Huschen	NLSSOL	NPOPT
DIST	2 3 3	2 3 3	3 4 4	1 2 2	1 2 2
BAZA	4 5 5	4 5 5	13 14 14	1 2 2	1 2 2
GMW	3 3 3	3 3 3	3 3 3	1 2 2	4 5 5
PAW	3 3 3	3 3 3	4 4 4	1 2 2	1 2 2
DAS1	1 2 2	1 2 2	1 2 2	1 2 2	1 2 2
DAS2	2 3 3	5 6 6	44 45 45	1 2 2	8 9 9
DAS3	2 3 3	5 6 6	75 76 76	1 2 2	7 8 8
DAS4	4 4 4	4 4 4	75 75 75	1 2 2	12 16 16
Bob8	1 2 2	1 2 2	1 2 2	1 2 2	1 2 2
TP001	6 8 8	4 5 5	19 23 23	30 51 51	19 22 22
TP002	21 13 13	20 12 12	8 8 8	14 24 24	11 16 16
TP028	1 2 2	1 2 2	1 2 2	1 2 2	10 11 11
TP032	5 4 4	5 4 4	5 4 4	1 2 2	2 3 3
TP048	1 2 2	1 2 2	1 2 2	1 2 2	8 9 9
TP049	10 11 11	10 11 11	15 16 16	25 26 26	42 44 44
TP050	6 7 7	6 7 7	8 9 9	10 11 11	14 17 17
TP051	1 2 2	1 2 2	1 2 2	1 2 2	7 8 8
TP052	1 2 2	1 2 2	1 2 2	1 2 2	3 5 5
TP053	1 2 2	1 2 2	1 2 2	1 2 2	5 6 6
TP224	2 3 3	2 3 3	4 5 5	2 3 3	4 5 5
TP231	20 26 26	25 38 38	18 22 22	26 41 41	16 21 21
TP269	1 2 2	1 2 2	1 2 2	1 2 2	5 6 6
TP354	21 22 22	10 11 11	9 10 10	13 14 14	18 26 26
WrHo1	12 15 15	17 22 21	137 138 138	18 28 28	51 67 67
WrHo2	30 50 50	23 39 33	152 154 154	—	64 82 82
RELN	1 2 2	1 2 2	1 2 2	3 4 4	3 4 4
Average	6 8 8	6 8 7	23 24 24	6 9 9	12 15 15
Failures	0	0	0	1	0

11 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 [42]. 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 [41]. 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 16. In current 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 16: 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 [28] 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 11 summary results are presented from a comparison between the same solvers as in the first test in Section 10. 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 [29].

Table 17: 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 [42].

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

12 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.

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 [25]. 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 [33]. Preliminary results are discussed in [25]. 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 [5].

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 [21].

Acknowledgements

I am grateful to Michael Saunders and Philip E. Gill for providing me with with access to all SOL solvers. Michael has furthermore helped me with numerical and algorithmic details. David M. Gay assisted in making the AMPL interface work. Yangquan Chen spent a lot of time helping me make the CUTE interface work on PC systems. Anders Forsgren has provided Fortran MEX-file interfaces, which has been the source of inspiration for the development of the C MEX-file interfaces. Arthur Jutan suggested using spline approximations for numerical differentiation. Donald R. Jones has been a source of inspiration for our global optimization work, and been very helpful in discussions concerning the implementation of the DIRECT and EGO algorithms. My graduate students Erik Dotzauer, Mattias Björkman and Jöran Petersson have made important contributions to TOMLAB. Many students and TOMLAB users have also contributed with bits and pieces, which I hereby acknowledge.

13 Conclusions

TOMLAB is an open and general optimization development environment in Matlab. Many internal algorithms are implemented, as well as interfaces to other algorithms in Matlab, Fortran and C. It is a flexible tool, with menu programs, a graphical user interface and driver routines for many types of optimization problems.

A large set of optimization problems are included as Matlab m-file code. It is easy for the user to add new sets of problems or solve single problems. TOMLAB may also run problems defined in the CUTE SIF language using DLL files or in the AMPL language (after making *nl*-files). TOMLAB is suitable for computer based learning in optimization courses, both algorithmic and applied, and in computer exercises.

The possibility to very easily use both automatic and numerical differentiation makes TOMLAB especially useful in practical applications, where derivative information may be hard to obtain. The global optimization routines are suitable for the common case of costly functions, e.g. parameter estimation in simulation models. The robust constrained nonlinear least squares solvers are efficient tools in the solution of applied parameter estimation problems. With the open design and the interfaces the Fortran solvers, CUTE and AMPL, there are many possible ways to utilize the system in the solution of applied optimization problems.

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] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear Programming and Network Flows*. John Wiley and Sons, New York, 2nd edition, 1990.

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

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

[7] I. Bongartz, A. R. Conn, N. I. M. Gould, and P. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[8] I. Bongartz, A. R. Conn, Nick Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. Technical report, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, September 2 1997.

[9] Mary Ann Branch and Andy Grace. *Optimization Toolbox User's Guide*. 24 Prime Park Way, Natick, MA 01760-1500, 1996.

[10] Thomas Coleman, Mary Ann Branch, and Andy Grace. *Optimization Toolbox User's Guide*. 24 Prime Park Way, Natick, MA 01760-1500, 1999. Third Printing Revised for Version 2 (Release 11).

[11] A. R. Conn, Nick Gould, A. Sartenaer, and Ph. 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.

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

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

[14] Roger Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, New York, 2nd edition, 1987.

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

[16] R. Fourer, D. M. Gay, and B. W.Kernighan. *AMPL - A Modeling Language for Mathematical Programming*. The Scientific Press, Redwood City, CA, 1993.

[17] David M. Gay. Hooking your solver to AMPL. Technical report, Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974, 1997.

[18] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, 1982.

[19] D. Goldfarb and M. J. Todd. Linear 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.

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

[21] Thomas 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.

[22] Kaj Holmberg. Heltalsprogrammering och dynamisk programmering och flöden i nätverk och kombinatorisk optimering. Technical report, Division of Optimization Theory, Linköping University, Linköping, Sweden, 1988-1993.

[23] 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.

[24] 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.

[25] 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.

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

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

[28] K. Holmström, Anders Ahnesjö, and Jöran Petersson. Algorithms for exponential sum fitting in radiotherapy planning. 2000. To be submitted.

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

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

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

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

[33] 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.

[34] 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.

[35] Donald R. Jones. DIRECT. *Encyclopedia of Optimization*, 1999. To be published.

[36] 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.

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

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

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

[40] 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.

[41] Jöran 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.

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

[43] A. Sartenaer. Automatic determination of an initial trust region in nonlinear programming. Technical Report 95/4, Department of Mathematics, Facultés Universitaires ND de la Paix, Bruxelles, Belgium, 1995.

[44] 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.