AMO - Advanced Modeling and Optimization Volume 1, Number 1, 1999

The TOMLAB Optimization Environment in Matlab[1]

Kenneth Holmström[2]
Center for Mathematical Modeling,
Department of Mathematics and Physics
Mälardalen University, P.O. Box 883, SE-721 23 Västerås, Sweden

Abstract

TOMLAB is a general purpose, open and integrated Matlab development environment for research and teaching in optimization on Unix 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.* In this paper we discuss the design and contents of TOMLAB, as well as some applications where TOMLAB has been successfully applied.

TOMLAB is based on NLPLIB TB, a Matlab toolbox for nonlinear programming and parameter estimation, and OPERA TB 1.0, a Matlab toolbox for linear and discrete optimization. More than 65 different algorithms and graphical utilities are implemented. It is possible to call solvers in the Matlab Optimization Toolbox and general-purpose solvers implemented in Fortran or C using a MEX-file interface. Currently, MEX-file interfaces have been developed for *MINOS*, *NPSOL*, *NPOPT*, *NLSSOL*, *LPOPT*, *QPOPT* and *LSSOL*. A problem is solved by a direct call to a solver or by a general multi-solver driver routine, or interactively, using a graphical user interface (GUI) or a menu system. If analytical derivatives are not available, automatic differentiation is easy using an interface to ADMAT/ADMIT TB. Furthermore, five types of numerical differentiation methods are included in NLPLIB TB.

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 currently solves small and medium size dense problems, where the available memory is the limit. TOMLAB is free for academic purposes and downloadable at our home page.

Keywords: Mathematical Programming, Nonlinear Programming, Operations Research, Matlab, CUTE, AMPL, Graphical User Interface, Software Engineering, Mathematical Software, Optimization, Algorithms, Nonlinear Least Squares.

AMS Subject Classification: 90C99, 90C30, 90C99

[1]Financed by the Mälardalen University Research Board, project *Applied Optimization and Modeling (TOM).*
[2]E-mail: **hkh@mdh.se**; URL: **http://www.ima.mdh.se/tom**.

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 one strong motivation for starting the development of TOMLAB [32, 31]; a general-purpose, open and integrated development environment in Matlab for research and teaching in optimization. Another motivation was to provide powerful tools for the research on the hard-to-solve applied problems our research group are working on, by systematic development of a large set of usable optimization tools. This paper presents the design and contents of TOMLAB.

To solve optimization problems, traditionally the user has been forced to write a Fortran code that calls some standard solver written as a Fortran subroutine. For nonlinear problems, the user must also write subroutines computing the objective function value and the vector of constraint function values. The needed derivatives are either explicitly coded, computed by using numerical differences or derived using automatic differentiation techniques.

In recent years several modeling languages are developed, like AIMMS [5], AMPL [21], ASCEND [49], GAMS [6, 10] and LINGO [1]. The modeling system acts as a preprocessor. The user describes the details of his problem in a very verbal language; an opposite to the concise mathematical description of the problem. The problem description file is normally modified in a text editor, with help from example files solving the same type of problem. Much effort is directed to the development of more user friendly interfaces. The model system processes the input description file and calls any of the available solvers. For a solver to be accessible in the modeling system, special types of interfaces are developed.

The modeling language approach is suitable for many management and decision problems, but may not always be the best way for engineering problems, which often are nonlinear with a complicated problem description. Until recently, the support for nonlinear problems in the modeling languages has been crude. This is now rapidly changing [16].

For people with a mathematical background, modeling languages often seems to be a very tedious way to define an optimization problem. There has been several attempts to find languages more suitable than Fortran or C/C++ to describe mathematical problems, like the compact and powerful APL language [39, 50]. Nowadays, languages like Matlab has a rapid growth of users. Matlab was originally developed [45] as a preprocessor to the standard Fortran subroutine libraries in numerical linear algebra, LINPACK [12] and EISPACK [53] [22], much the same idea as the modeling languages discussed above. Matlab of today is an advanced and powerful tool, with graphics, animation and advanced menu design possibilities integrated with the mathematics. The Matlab language has made the development of toolboxes possible, which serves as a direct extension to the language itself. Using Matlab as an environment for solving optimization problems offers much more possibilities for analysis than just the pure solution of the problem.

The concept of TOMLAB is to integrate all different systems, getting access to the best of all worlds. Furthermore TOMLAB provides well-implemented solution algorithms for most common optimization problems. TOMLAB should be seen as a complement to existing model languages, for the user needing more power and flexibility than given by a modeling system.

Much effort has been put on advanced design utilizing the power and features of MatlabṪhe 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.

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. 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 Sweden TOMLAB has been used for several years in optimization courses. It is distributed free of charge for academic purposes. During the first months since the first official release of TOMLAB, December 11, 1998, about 800 researchers and research groups have downloaded the software. All users have been requested to answer what applications they have in mind for TOMLAB, and it is clear that most fields of applied mathematics are represented. So there is an obvious need for good optimization tools in the scientific community.

This paper is organized as follows. In Section 2 we discuss the design of TOMLAB. Overviews of the two toolboxes that are part of TOMLAB are given in Section 3 (NLPLIB TB [29]) and Section 4 (OPERA TB [29]). The graphical user interface (GUI) is further discussed in [15, 14]. The MEX-file interfaces to several solvers are discussed in Section 5, and the interface to the Matlab Optimization Toolbox [9] is discussed in Section 6. Currently TOMLAB may run problems defined in CUTE [8] (Section 7) and AMPL [21] (Section 8). Finally, we end with some conclusions in Section 10.

2 The Design of TOMLAB

As the scope of TOMLAB is large and broad, there is a clear need of a well-designed system. It is also necessary to use the power of the Matlab language, to make the system flexible and easy to use and maintain. We have used the concept of structure arrays and made use of both the ability in Matlab to execute Matlab code defined as string expressions and to execute functions specified by a string. To be able to live up to the basic design principle of TOMLAB, *define your problem once, optimize using any suitable solver*, very many different interface routines, as well as several interface layers was needed. It is a difficult task to provide the necessary information for a particular solver, especially when the problem class is just a subset of the class of problems the particular solver is able to handle. To solve this design problem in a general and easily expandable way, we use one structure array that holds all information about the problem, all input for the solution process. In a similar way one structure array holds the all output from the optimization, together with the input structure array. This makes advanced post-processing of results easy.

The thought is that the routines in TOMLAB may be put together into more advanced hybrid algorithms or being subparts in more complex algorithms. Therefore it is important to organize all handling of global information, to avoid conflicts. We use a stack strategy for the global variables, that works well when making recursive calls in separable nonlinear least squares algorithms. This algorithm is used to solve the chemical equilibrium analysis problems we currently are working with.

The size of the system is quite large; only the Matlab code consists of more than 400 m-files and more than 68 000 lines of code, and it is rapidly growing. This motivates a well-defined naming convention and design. In the following we describe this design, which is fully developed in NLPLIB TB and partly (for linear programming) in OPERA TB.

TOMLAB solves a number of different types of optimization problems. Currently, we have defined the types listed in Table 1. The global variable *probType* is the current type to be solved. 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. For example, a constrained nonlinear programming solver should in principle be able to solve nearly all types of problems defined.

Define *probSet* to be a set of defined optimization problems to be solved. Each *probSet* belongs to a certain class of optimization problems of type *probType*. In Table 2 the currently defined problem sets are listed, and new sets are easily added. The *probSet* **usr** is defined in order to make the inclusion of a few optimization problems of any type a simple and fast task. This method is to prefer when TOMLAB is used in optimization courses.

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

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

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

Table 2: Defined test problem sets in TOMLAB.

probSet	probType	Description of test problem set
uc	1	Unconstrained test problems.
qp	2	Quadratic programming test problems.
con	3	Constrained test problems.
ls	4	Nonlinear least squares test problems.
exp	5	Exponential model fitting problems.
cls	6	Linear constrained nonlinear least squares problems.
nls	6	Nonlinear constrained nonlinear least squares problems.
glb	9	Box-bounded global optimization test problems.
glc	10	Global MINLP test problems.
mgh	4	More, Garbow, Hillstrom nonlinear least squares problems.
amp	3	AMPL test problems as *nl*-files.
cto	3	CUTE constrained test problems as *dll*-files.
ctl	3	CUTE large constrained test problems as *dll*-files.
uto	1	CUTE unconstrained test problems as *dll*-files.
utl	1	CUTE large unconstrained test problems as *dll*-files.
nts	7	Nonlinear time series.
usr	1-9	User defined problems of *probType* 1-9.

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 *Interface Routines* in Figure 1 are used to convert computational results to the form expected by different solvers. 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. The *Gateway Routines* used for nonlinear least squares also implements different types of weighting schemes as well as separable nonlinear least squares algorithms.

It was necessary to make a layer with *Gateway Routines* to make the code of the *Low Level Routines* clean enough to make automatic differentiation possible.

Not shown in Figure 1 is the possibility to directly call the TOMLAB solvers, if an input structure is first created. Tools to easily create this structure is provided.

A large set of Matlab m-files are needed to implement the chain of function calls for all solver types and problem sets. For the menu systems, driver routines etc. Table 3 shows the naming convention used, e.g. when (*probType* = **con**, routines *conOpt*, *conRun*, *conDef* and *conSolve* are available.

The names of the problem setup routine and the low level routines are constructed as two parts. The first part being the abbreviation used in Table 2 for the *probSet*. The second part denoting the computed task, as shown in Table 4. An example of the process of optimization and the function names for the constrained nonlinear programming case (*solvType* = **con**, *probSet* = **con**) is shown in Figure 2.

Table 3: Names of main m-file functions in NLPLIB TB.

Generic variable	**Purpose** (*solvType* is $\diamond$, e.g. $\diamond$=**con**)
$\diamond$Opt	Menu program.
$\diamond$Run	Multi-solver optimization driver routine.
$\diamond$Def	Routine defining optimization parameters.
$\diamond$Solve	(Prototype) solver.

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

Generic name	**Purpose** ($\diamond$ is any *probSet*, e.g. $\diamond$=**amp**)
$\diamond$_prob	Define string matrix with problems and a structure *prob* for each 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 2nd part of 2nd 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 2nd part of the Hessian matrix, $\sum_i r_i(x) \partial^2 r_i(x)/dx^2$

The problem setup routine has two modes of operation; either return a **string matrix** with the names of the problems in the *probSet* or a structure with all information about the selected problem. The structure, named *Prob*, is shown in Table 5, and its subtables. We refer to the User's Guide [34] for a full description of all different fields. 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 be solved.

There are default values for everything that is possible to set defaults for, and all routines are written in a way that makes it possible for the user to just set an input argument empty and get the default. Automatic differentiation is easy using an interface to ADMAT/ADMIT TB and five types of numerical differentiation are included. The fields *AutoDiff* and *NumDiff* in the *Prob* structure controls the use of differentiation method. If routines for derivatives are non-present, numerical differentiation is automatically used by the solvers. More information about ADMAT TB is found at the URL: **http://simon.cs.cornell.edu/home/verma/AD/**.

The results of the optimization attempts are stored in a structure array named *Result*. The currently defined fields in the structure are shown in Table 15. The use of structure arrays make advanced result presentation and statistics possible.

The field *xState* describes the state of each of the variables. In Table 16 the different values are described. The different conditions for linear constraints are defined by the state variable in field *bState*. In Table 17 the different values are described.

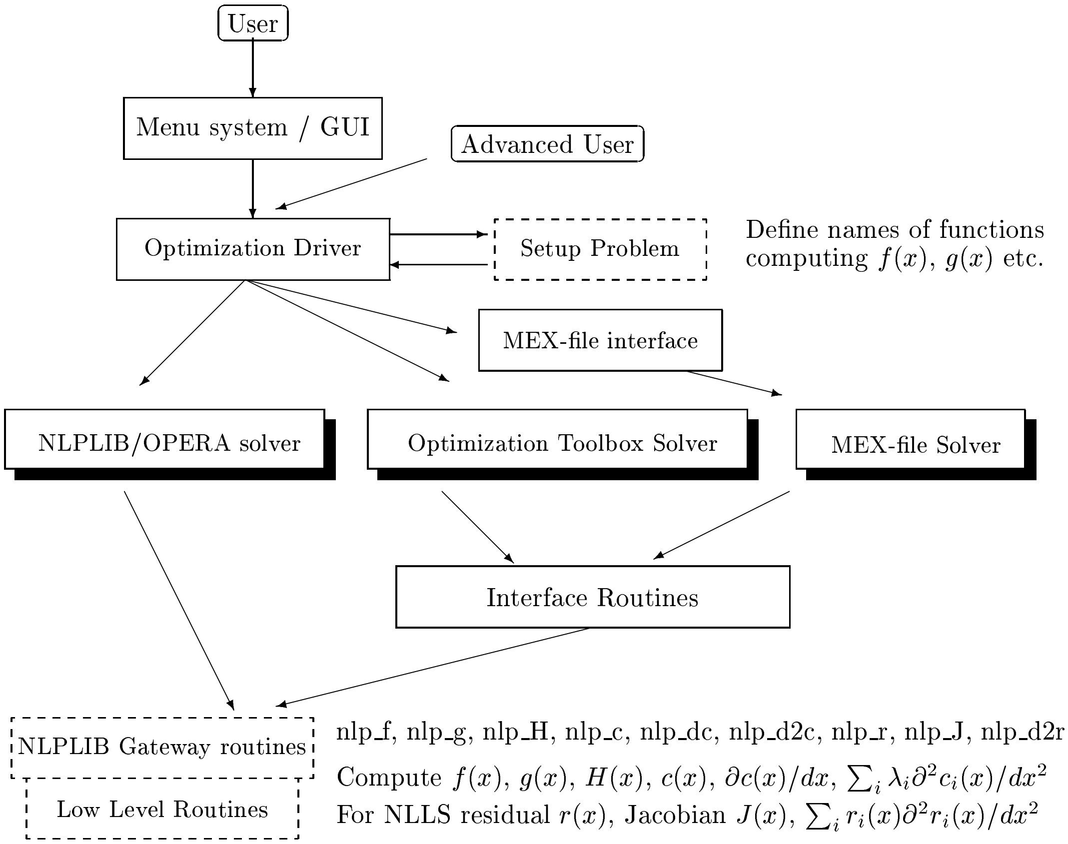

Figure 1: The process of optimization in TOMLAB.

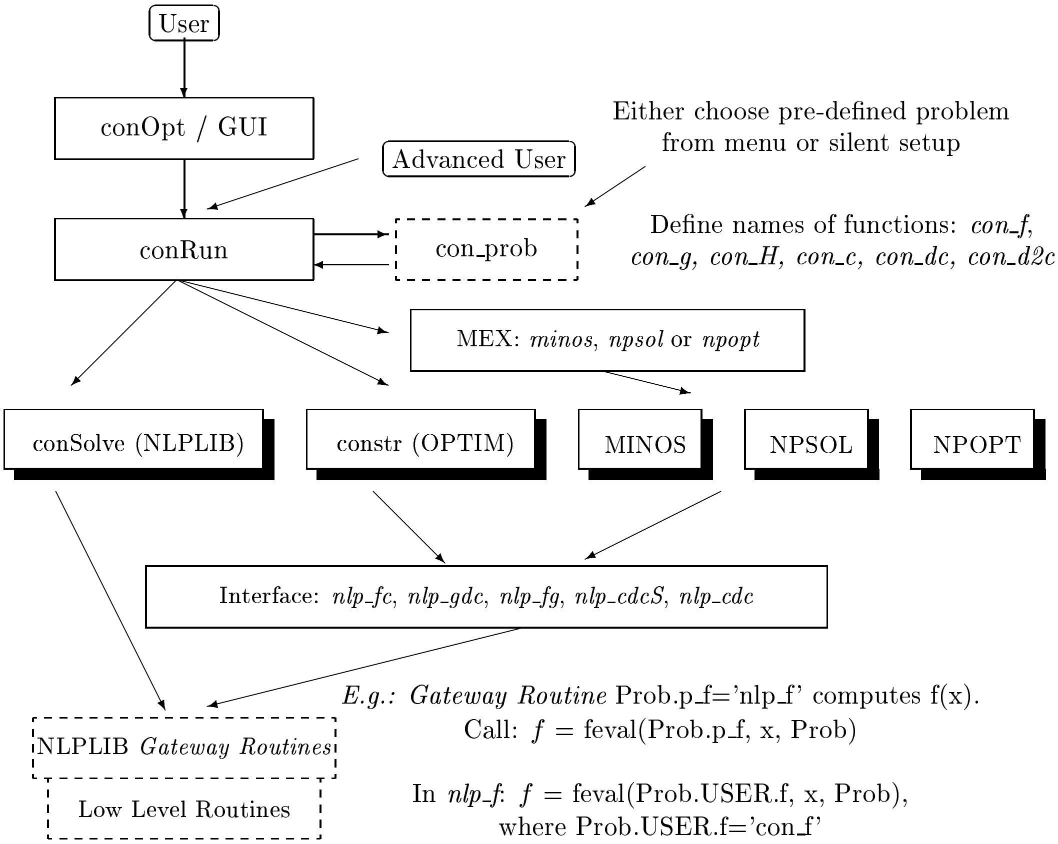

Figure 2: Solution of constrained nonlinear problems in TOMLAB.

To conclude, the system design is flexible and easy to expand in many different ways.

3 The NLPLIB Toolbox

The NLPLIB TB is a set of Matlab m-files, which implements optimization solvers, test problems and utilities for nonlinear optimization and nonlinear parameter estimation. The focus is on dense problems. There are eight main fields; unconstrained and constrained optimization, quadratic programming, box-bounded global optimization, global mixed-integer nonlinear programming, unconstrained and constrained nonlinear least squares and exponential sum model data fitting. There are four ways to solve a predefined problem: defining a problem input structure and calling the solver algorithm, calling a multi-solver driver routine, using a menu system or running the Graphical User Interface (GUI). NLPLIB TB is described in more detail in [34]. Also see the User's Guide for TOMLAB [36], which is up-dated regularly and available on-line.

Currently NLPLIB TB 1.0 consists of about 54000 lines of m-file code in 300 files with algorithms, utilities and predefined problems. Included are new algorithms for the nonlinear parameter estimation problem of fitting sums of exponential functions to empirical data. The TOMLAB solvers all explicitly handle bounds and linear constraints, with an input model upper/lower bound format like NPSOL.

The routine *ucSolve* implements a prototype algorithm for unconstrained optimization with simple bounds on the parameters. It includes several of the most popular search step methods for unconstrained optimization. Bound constraints are treated as described in Gill et al.[25]. The search step methods for **unconstrained optimization** are: 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. For the Newton and the quasi-Newton methods the code is using a subspace minimization technique to handle rank problems, see Lindström [43]. The quasi-Newton codes also use safe guarding techniques to avoid rank problem in the updated matrix.

For general **nonlinear problems with nonlinear constraints** a sequential quadratic programming (SQP) method by Schittkowski [52] is implemented in the routine *conSolve*, as well as an Han-Powell SQP method. The routine *nlpSolve* implements the Filter SQP by Fletcher and Leyffer [19]. Another constrained solver in NLPLIB TB is the structural trust region algorithm implemented in *sTrustR*, combined with an initial trust region radius algorithm. The code is based on the algorithms in [11] and [51], and treats partially separable functions. The constraints should be convex for the routine to work.

Quadratic programs (qp) are solved with a standard active-set method [44], implemented in the routine *qpSolve*.

The prototype nonlinear least squares algorithm *lsSolve* treats problems with bound constraints in a similar way as the routine *ucSolve*. The prototype routine *lsSolve* implements four search methods for nonlinear least squares problems: the Gauss-Newton method, the Al-Baali-Fletcher [3] and the Fletcher-Xu [20] hybrid method, and the Huschens TSSM method [37]. If rank problems occur *lsSolve* uses subspace minimization.

The constrained nonlinear least squares solver *clsSolve* is based on *lsSolve* and its search steps methods. Currently *clsSolve* treats linear equality and inequality constraints using an active-set strategy.

The line search algorithm *LineSearch* used by the solvers *conSolve*, *lsSolve*, *clsSolve* and *ucSolve* is a modified version of the algorithm by Fletcher [18, chap. 2]. The use of quadratic and cubic interpolation is possible in the line search algorithm. For more details, see [36].

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. problems that have finite simple bounds on all the variables. *glbSolve* implements the DIRECT algorithm [41], 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**), *glcSolve* implements an extended version of DIRECT, see [40]. For global optimization problems with expensive function evaluations the routine *ego* implements the Efficient Global Optimization (EGO) algorithm [42].

4 The OPERA Toolbox

The Matlab toolbox OPERA TB is a collection of Matlab m-files which solves many basic linear and discrete optimization problems in operations research and mathematical programming. Currently OPERA TB 1.0 consists of about 15 000 lines of Matlab m-file code in 57 files with algorithms and utilities and 45 example files. OPERA TB was originally presented in [30] and the current version is presented in [35]. Also see the User's Guide for TOMLAB [36], which is up-dated regularly and available on-line.

There are several algorithms implemented for **linear programs** (LP), the standard revised simplex algorithm and two polynomial algorithms. The dual simplex method, usable when a dual feasible solution is available instead of a primal feasible, is also implemented.

For **transportation problems** the transportation simplex method is included, together with three simple algorithms to find a starting basic feasible solution.

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]. Implemented are routines for network search and finding the minimal spanning tree. There are three algorithms to solve the shortest path problem, an algorithm to solve maximum flow problems using the Ford-Fulkerson augmenting path method, and a network simplex algorithm to solve the minimum cost network flow problem. The symmetric traveling salesman problem is solved using Lagrangian relaxation and the subgradient method with the Polyak rule II, an algorithm by Held and Karp [26].

To solve mixed linear **integer programs** a branch and bound algorithm from Nemhauser and Wolsey [47, chap. 8] and a cutting plane algorithm using Gomory cuts are implemented. Balas method for 0/1 integer programs restricted to integer coefficients is also implemented.

Included in the toolbox are also some examples of dynamic programming and Lagrangian relaxation techniques. For linear programs a menu system and a multi-solver driver routine is implemented in the same way as in NLPLIB TB.

For linear and integer programming both classroom solvers as well as more robust solvers are available.

5 The MEX-file Interface

TOMLAB is an open system with possibilities to interact with other program packages. An optimization solver implemented in Fortran or C is called from TOMLAB using a MEX-file interface. MEX-file interfaces for both Fortran and C are easy to develop for Unix systems. Interfaces to many solvers are available on Unix. On PC machines, there has been problems to make Fortran MEX-file interfaces that work properly. We have developed general MEX-file interfaces in C and converted solvers written in Fortran to C using the Fortran to C converter *f2c* [17]. This solution is well-working and it should be easy to expand the list of available solvers to TOMLAB.

Presently, MEX-file interfaces have been developed for six general-purpose solvers available from the Systems Optimization Laboratory, Department of Operations Research, Stanford University, California; NPSOL 5.02 [24], NPOPT 1.0-10 (updated version of NPSOL), NLSSOL 5.0-2, QPOPT 1.0-10, LSSOL 1.05 and LPOPT 1.0-10. Furthermore, an interface to MINOS 5.5 [46] has been developed. MEX-file interfaces are available for both Unix and PC systems.

6 The Matlab Optimization Toolbox Interface

Included in TOMLAB is an interface to several of the solvers in the Matlab Optimization Toolbox (OPTIM) [9]. The solvers callable are listed in Table 19. The TOMLAB optimization driver routine checks if the routine is in the path and then calls the Matlab function **feval** to evaluate it. Two low-level interface routines have been written. The *constr* solver needs both the objective function and the vector of constraint functions in the same call, which *nlp_fc* supplies. Also the gradient vector and the matrix of constraint normals should be supplied in one call. These parameters are returned by the routine *nlp_gdc*.

7 The CUTE Interface

The Constrained and Unconstrained Testing Environment (CUTE) [8, 7] 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 in NLPLIB TB, using any solver callable from the toolbox.

CUTE stores the problems in the standard input format (SIF) in files with extension *sif*. There are tools to select appropriate problems from the data base. Running CUTE, a SIF decoder creates up to five Fortran files; *elfuns, extern, groups, ranges*, and *settyp*, and one ASCII data file; *outsdif.dat* or *outsdif.d*. The Fortran files are compiled and linked together with the CUTE library and a solver routine. Running the binary executable, the problem is solved using the current solver. During the solution procedure, the ASCII data file *outsdif.dat* or *outsdif.d* is read.

With the CUTE distribution follows a Matlab interface. There are one MEX-file interface routine *ctools.f* for constrained CUTE problems, and one MEX-file interface routine *utools.f* for unconstrained problems. The MEX-file interface routine is compiled and linked together with the Fortran files, generated by the SIF decoder, and the Matlab MEX library to make a DLL (Dynamic Link Library) file. At run-time, Matlab calls the DLL, which will read the CUTE ASCII data file for the problem specific information. Also included in the CUTE distribution is a set of Matlab low-level m-file interface routines that calls the appropriate MEX-file interface routine.

For the TOMLAB CUTE interface we assume that the DLL files are already built and stored in predefined directories. The name of the dll is the problem name used by CUTE, e.g. *rosenbr.dll* for the Rosenbrock banana function. The ASCII data file also has a unique name, e.g. *rosenbr.dat*. The CUTE Matlab low-level m-file interface routines assumes the DLL file to be named ctools.dll and utools.dll (and the data file to be called outsdif.dat on PC). TOMLAB calls the Matlab files in the CUTE distribution, but to solve the name conflict, uses special m-files *ctools.m* and *utools.m* to make a call to the correct DLL file. The ASCII data file must also be copied to a temporary file, with the necessary filename *outsdif.dat*, before executing the DLL.

When using the TOMLAB interface, the user either gets a menu of all DLL files in the CUTE directory chosen, or directly makes a choice of which problem to solve. Precompiled DLL files for the CUTE data set is available, or the necessary files for the user to build his own DLL files.

8 The AMPL Interface

Using interfaces between a modeling language and TOMLAB could be of great benefit and improve the possibilities for analysis on a given problem. As a first attempt, a TOMLAB interface to the modeling language AMPL [21] was built. The reason to choose AMPL was that it has a rudimentary Matlab interface written in C [23] that could easily be used.

AMPL is using ASCII files to define a problem. The naming convention is to use the problem name and various extensions, e.g. *rosenbr.mod* and *rosenbr.dat* for the Rosenbrock banana function.

These files are normally converted to binary files with the extension *nl*, called *nl*-files, e.g. *rosenbr.nl*. After solving the problem, the solver creates a file with extension *sol*, e.g. *rosenbr.sol*, containing a termination message and the solution it has found.

The current TOMLAB AMPL interface is an interface to the problems defined in the AMPL *nl*-format. When using the TOMLAB interface, the user either gets a menu of the *nl*-files found or directly makes a choice of which problem to solve. The initialization routine in TOMLAB for AMPL problems, *amp_prob*, either calls *amplfunc* or *spamfunc*, the two MEX-file interface routines written by Gay [23]. The TOMLAB low level routines *amp_f, amp_g,* etc. calls the same MEX-file interface routines, and dependent on the parameters in the call, the appropriate information is returned.

Note that the design of the AMPL solver interface makes it easy to run the NLPLIB TB solvers from AMPL using the Matlab Engine interface routines, a possible extension in the future. But indeed, any solver callable from NLPLIB TB may now solve problems formulated in the AMPL language.

9 Applications of TOMLAB

TOMLAB has already 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, we have formulated a separable nonlinear least squares algorithm. 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 [33]. 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. Results so far are excellent and *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 for more than ten years [38]. Preliminary results are discussed in [33]. 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 Prof. 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 [28].

TOMLAB has also been an essential tool in the applied projects researched in the licentiate thesis of my graduate students Erik Dotzauer (Algorithms for Short-Term Production Planning of Cogeneration Plants [13]), Jöran Petersson (Algorithms for Fitting Two Classes of Exponential Sums to Data [48]) and Thomas Hellström (A Randow Walk through the Stock Market [27]).

10 Conclusions and Future work

TOMLAB is an open and general optimization development environment in Matlab. Many internal algorithms are implemented in the two toolboxes NLPLIB TB and OPERA TB, 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 multi-solver 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 code new problems using the predefined files as a model. 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.

TOMLAB is a powerful tool, both in the development of general-purpose optimization software and in the solution of applied optimization problems.

Future plans include extension of TOMLAB to sparse optimization problems, interface to the new routines in MathWorks Optimization Toolbox 2.0, MEX-file interfaces to more solvers, like CFSQP, and interfaces to the commercial modeling systems GAMS and XPRESS-MP.

Acknowledgements

I would like to thank Prob. Philip E. Gill for giving me access to the commercial codes NPSOL, NPOPT, NLSSOL, LPOPT, QPOPT and LSSOL. I am grateful to Prof. Michael Saunders for sending me the MINOS distribution and his continuous support of the TOMLAB development. Prof. David M. Gay helped making the AMPL interface work. I would also thank Prof. Yangquan Chen for a lot of help and hard work in making the CUTE interface run on PC systems. Prof. Anders Forsgren has provided Fortran MEX-file interfaces, which has been the source of inspiration for the development of the C MEX-file interfaces. Prof. Arthur Jutan has given several good suggestions concerning numerical differentiation. Dr. Arun Verma has kindly supplied improved versions of his ADMAT TB to cope with the trouble of automatic differentiation for the different TOMLAB problems. Many others around the world has given us interesting suggestions, found bugs and discussed the system, and we hereby thank all people that have contributed.

All our efforts on global optimization have only been possible because of Dr. Donald R. Jones, which has supplied us with preprints and papers, and advice on implementation issues.

My graduate students, Mattias Björkman, Erik Dotzauer and Jöran Petersson have made many important contributions to TOMLAB. Many students have also contributed with bits and pieces, which I hereby acknowledge.

References

[1] *LINGO - The Modeling Language and Optimizer*, LINDO Systems Inc., Chicago, IL, 1995.

[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 (1985), pp. 405–421.

[4] J. M. Berg and K. Holmström, *On Parameter Estimation Using Level Sets*, SIAM Journal on Control and Optimization, (1999). To be published.

[5] J. Bisschop and R. Entriken, *AIMMS - The Modeling System*, Paragon Decision Technology, Haarlem, The Netherlands, 1993.

[6] J. Bisschop and A. Meeraus, *On the development of a general algebraic modeling system in a strategic planning environment*, Mathematical Programming Study, 20 (1982), pp. 1–29.

[7] I. Bongartz, A. R. Conn, N. Gould, and P. L. Toint, *CUTE: Constrained and Unconstrained Testing Environment*, tech. rep., IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, September 2 1997.

[8] 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 (1995), pp. 123–160.

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

[10] A. Brooke, D. Kendrick, and A. Meeraus, *GAMS - A User's Guide*, The Scientific Press, Redwood City, CA, 1988.

[11] A. R. Conn, N. 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 (1996), pp. 1059–1086.

[12] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK User's Guide*, SIAM, (1979).

[13] E. Dotzauer, *Algorithms for Short-Term Production-Planning of Cogeneration Plants*, Licentiate Thesis No. 644, ISBN 91-7219-033-7, ISSN 0280-7971, Division of Optimization, Department of Mathematics, Linköping University, Linköping, Sweden, 1997.

[14] E. Dotzauer and K. Holmström, *A Graphical User Interface for Nonlinear Programming in Matlab*, Annals of Operations Research, Modeling Languages and Approaches (1998). Submitted.

[15] ——, *The TOMLAB Graphical User Interface for Nonlinear Programming*, Advanced Modeling and Optimization, 1 (1999).

[16] A. S. Drud, *Interactions between nonlinear programing and modeling systems*, Mathematical Programming, Series B, 79 (1997), pp. 99–123.

[17] S. I. Feldman, D. M. Gay, M. W. Maimone, and N. L. Schryer, *A Fortran-to-C converter*, Tech. Rep. Computing Science Technical Report No. 149, AT&T Bell Laboratories, May 1992.

[18] R. Fletcher, *Practical Methods of Optimization*, John Wiley and Sons, New York, 2nd ed., 1987.

[19] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Tech. Rep. NA/171, University of Dundee, 22 September 1997.

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

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

[22] B. S. Garbow, J. M. Boyle, J. J. Dongara, and C. B. Moler, *Matrix Eigensystem Routines-EISPACK Guide Extension*, in Lecture Notes in Computer Science, Springer Verlag, New York, 1977.

[23] D. M. Gay, *Hooking your solver to AMPL*, technical report, Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974, 1997.

[24] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *User's Guide for NPSOL (Version 4.0): A Fortran package for nonlinear programming*, Department of Operations Research, Stanford University, Stanford, CA, 1986. SOL 86-2.

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

[26] M. Held and R. M. Karp, *The Traveling-Salesman problem and minimum spanning trees: Part II*, Mathematical Programming, 1 (1971), pp. 6–25.

[27] T. Hellström, *A Random Walk through the Stock Market*, Licentiate Thesis, ISSN-0348-0542, UMINF 98.16, Department of Computing Science, Umeå University, Sweden, December 10 1998.

[28] T. Hellström and K. Holmström, *Parameter Tuning in Trading Algorithms using ASTA*, in Computational Finance – Proceedings of the Sixth International Conference, Leonard N. Stern School of Business, January 1999, Y. S. Abu-Mostafa, B. LeBaron, A. W. Lo, and A. S. Weigend, eds., Cambridge, MA, 1999, MIT Press.

[29] K. Holmström, *NLPLIB TB 1.0 - A Matlab Toolbox for Nonlinear Optimization and Parameter Estimation*, Technical Report IMa-TOM-1997-02, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997.

[30] ——, *OPERA TB 1.0 - A Matlab Toolbox for Optimization Algorithms in Operations Research*, Technical Report IMa-TOM-1997-01, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997.

[31] ——, *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.

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

[33] ——, *Constrained Separable NLLS Algorithms for Chemical Equilibrium Analysis*, in Proceedings from the 5th Meeting of the Nordic Section of the Mathematical Programming Society, A. Løkketangen, ed., ISBN 82-90347-76-6, Molde, 1998, Division of Operations Research, Molde University.

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

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

[36] ——, *TOMLAB v1.0 User's Guide*, Technical Report IMa-TOM-1999-01, Department of Mathematics and Physics, Mälardalen University, Sweden, 1999.

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

[38] 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 (1996), pp. 717–734.

[39] K. Iverson, *A Programming Language*, John Wiley and Sons, New York, 1962.

[40] D. R. Jones, *DIRECT*, Encyclopedia of Optimization, (1999). To be published.

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

[42] D. R. Jones, M. Schonlau, and W. J. Welch, *Efficient global optimization of expensive Black-Box functions*, Journal of Global Optimization, 13 (1998), pp. 455–492.

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

[44] D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley Publishing Company, Reading, Massachusetts, 2nd ed., 1984.

[45] C. B. Moler, *MATLAB — An Interactive Matrix Laboratory*, Technical Report 369, Department of Mathematics and Statistics, University of New Mexico, 1980.

[46] B. A. Murtagh and M. A. Saunders, *MINOS 5.4 USER'S GUIDE*, Technical Report SOL 83-20R, Revised Feb. 1995, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305-4022, 1995.

[47] G. L. Nemhauser and L. A. Wolsey, *Integer programming*, in Optimization, G. L. Nemhauser, A. H. G. R. Kan, and M. J. Todd, eds., vol. 1 of Handbooks in Operations Research and Management Science, Elsevier/North Holland, Amsterdam, The Netherlands, 1989.

[48] 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.

[49] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg, *ASCEND: An object-oriented computer environment for modeling and analysis: The modeling language*, Computers and Chemical Engineering, 15 (1991), pp. 53–72.

[50] R. P. Polivka and S. Pakin, *APL: The Language and Its Usage*, Prentice Hall, Englewood Cliffs, N. J., 1975.

[51] 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.

[52] 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.

[53] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines - EISPACK Guide Lecture Notes in Computer Science*, Springer-Verlag, New York, 2nd ed., 1976.

Table 5: Information stored in the problem structure *Prob*.

Field	**Description**
Name	Problem name.
P	Problem number.
probType	TOMLAB problem type, see Table 1.
probFile	Name of m-file in which problem are defined.
xName	Name of each decision variable.
cName	Name of each general constraint.
optParam	Structure with special fields for optimization parameters, see Table 6.
Solver	Structure with fields *Name* and *Alg*. *Name* is the name of the solver and *Alg* is the solver algorithm to be used.
uP	User supplied parameters for the problem.
uPName	Problem name connected to the user supplied parameters.
ExpFit	Structure with special fields for exponential model fitting problems, see Table 7.
QP	Structure with special fields for quadratic problems, see Table 8.
NLLS	Structure with special fields for nonlinear least squares, see Table 9.
NTS	Structure with special fields for nonlinear time series, see Table 10.
PartSep	Structure with special fields for partially separable functions, see Table 11.
GLOBAL	Structure with special fields for global optimization, see Table 12.
A	Constraint matrix for linear constraints, one constraint per row.
b_L	Lower bounds on the linear constraints.
b_U	Upper bounds on the linear constraints.
c_L	Lower bounds on the general constraints.
c_U	Upper bounds on the general constraints.
x_L	Lower bounds on the variables.
x_U	Upper bounds on the variables.
x_0	Starting point.
N	Problem dimension (number of variables).
f_Low	Lower bound on function value. Used in line search by Fletcher, default, $-realmax = -1.7977E308$.
x_opt	Optimal point x^* (if known).
f_opt	Optimal objective function value $f(x^*)$.
AutoDiff	If true, use automatic differentiation.
NumDiff	Numerical approximation of derivatives. If set to 1, classical approach with forward or backward differences together with automatic step selection will be used. If set to 2, 3 or 4 the spline routines *csapi*, *csaps* or *spaps* in SPLINE Toolbox will be used. If set to 5, derivatives will be estimated by use of complex variables.
p_f	Name of Gateway routine computing the objective function $f(x)$.
p_g	Name of Gateway routine computing the gradient vector $g(x)$.
p_H	Name of Gateway routine computing the Hessian matrix $H(x)$.
p_c	Name of Gateway routine computing the vector of constraint functions $c(x)$.
p_dc	Name of Gateway routine computing the matrix of constraint normals $\partial c(x)/dx$.
p_d2c	Name of Gateway routine computing the 2nd part of 2nd derivative matrix of the Lagrangian function, $\sum_i \lambda_i \partial^2 c(x)/dx^2$.
p_r	Name of Gateway routine computing the residual vector $r(x)$.
p_J	Name of Gateway routine computing the Jacobian matrix $J(x)$.
p_d2r	Name of Gateway routine computing the second part of the Hessian for a nonlinear least squares problem, i.e. $\sum_{i=1}^{m} r_i(x) \frac{\partial^2 r_i(x)}{\partial x_j \partial x_k}$.
USER	Structure with user defined names of the m-files computing the objective, gradient, Hessian etc. See Table 13. These routines are called from the corresponding Gateway Routine
x_min	Lower plot region parameters.
x_max	Upper plot region parameters.

Table 6: Information stored in the structure *Prob.optParam*

Field	Description
alg	Optimization Algorithm. Dependent on type of problem. Default 0.
method	Solver sub-method technique. Default 0.
PriLev	Print level in optimization solver, default 1.
eps_x	Convergence tolerance in optimal solution x, distance between successive x, $\|x_{k+1} - x_k\|$, default 10^{-8}.
eps_f	Convergence tolerance on f. Also used when testing on the directed derivative, default 10^{-8}.
eps_dirg	Convergence tolerance on the directed derivative, default 10^{-8}.
eps_c	Constraint violation convergence tolerance, default 10^{-6}.
LineAlg	Line search algorithm. 0 = quadratic interpolation, 1 = cubic interpolation, Default $LineAlg = 0$.
GradCheck	Set to 1 if you want to check user-supplied gradients, default 0.
MaxIter	Maximum number of iterations, default 500.
DiffGradMinChange	Minimum change in variables for finite difference gradients, default 10^{-8}.
DiffGradMaxChange	Maximum change in variables for finite difference gradients, default 0.1.
InitStepLength	Initial step length, default 1 or less.
eps_g	Gradient (or reduced gradient) convergence tolerance, default 10^{-6}.
eps_Rank	Rank test tolerance, default 10^{-10}.
wait	Flag if to use pause statements after output, default 0.
eps_absf	Convergence tolerance on absolute function value, default *realmin*.
PreSolve	Flag if presolve analysis is to be applied on linear constraints, default 0.
QN_InitMatrix	Initial matrix for Quasi-Newton, may be set by the user. When *QN_InitMatrix* is empty, the identity matrix is used.
LineSearch	Structure with special fields for the line search, see Table 14.
Penalty	Penalty parameter for constrained problems.
xTol	If $x \in [x\_L, x\_L + bTol]$ or $[x\_U - bTol, x\_U]$, fix x on bound.
bTol	Feasibility tolerance for linear constraints.
cTol	Feasibility tolerance for nonlinear constraints.
fTol	Accuracy in the computation of the function value, default $eps^{0.9}$.
size_x	Size at optimum for the variables x, used in the convergence tests. Default 1.
size_f	Size at optimum for the function f, used in the convergence tests. Default 1.
size_c	Size at optimum for the constraints c, used in the convergence tests. Default 1.
LowIts	Number of iterations with low reduction before convergence.
NOT_release_all	Set to 1 if not to release more than one variable at the time.
subalg	Optimization sub algorithm. Dependent on type of problem. Default 0.
splineSmooth	Smoothness parameter sent to the SPLINE Toolbox routine *csaps.m* when computing numerical approximations of the gradient and the Jacobian. Default 0.4.
splineTol	Tolerance parameter sent to the SPLINE Toolbox routine *spaps.m* when computing numerical approximations of the gradient and the Jacobian. Default 10^{-3}.

Table 7: Information stored in the structure *Prob.ExpFit*

Field	**Description**
p	Number of exponential terms, default 2.
wType	Weighting type, default 1.
eType	Type of exponential terms, default 1.
infCR	Information criteria for selection of best number of terms, default 0.
dType	Differentiation formula, default 0.
geoType	Type of equation, default 0.
qType	Length q of partial sums, default 0.
sigType	Sign to use in $(P \pm \sqrt{Q})/D$ in *exp_geo* for $p = 3, 4$, default 0.
lambda	Vector of dimension p, intensities.
alpha	Vector of dimension p, weights.
x0Type	Type of starting value algorithm.
sumType	Type of exponential sum.
t_eqdist	Flag if data is equidistant in time.

Table 8: Information stored in the structure *Prob.QP*

Field	**Description**
F	Constant matrix, the Hessian
c	Constant vector.
B	Logical vector of the same length as the number of variables. A one corresponds to a variable in the basis.

Table 9: Information stored in the structure *Prob.NLLS*

Field	**Description**
weightType	Weighting type: *0* No weighting. *1* Weight with data in *Yt*. If $Yt = 0$, the weighting is 0, i.e. deleting this residual element. *2* Weight with weight vector or matrix in *weightY*. If *weightY* is a vector then weighting by $weigthY.*r$ (elementwise multiplication). If *weightY* is a matrix then weighting by $weigthY * r$ (matrix multiplication). *3* *nlp_r* calls the routine *weigthtY* (must be a string with the routine name) to compute the residuals.
weightY	Either empty, a vector, a matrix or a string, see *weightType*.
t	Time vector t.
Yt	Matrix with observations $Y(t)$.
UseYt	If $UseYt = 0$ compute residual as $f(x,t) - Y(t)$ (default), otherwise $Y(t)$ should be treated separately and the residual routines just return $f(x,t)$.
SepAlg	If $SepAlg = 1$, use separable non linear least squares formulation, default 0.

Table 10: Information stored in the structure *Prob.NTS*

Field	**Description**
SepAlg	If $SepAlg = 1$, use separable non linear least squares formulation, default 0.
ntsModel	Nonlinear model number
p	The number of terms (lags) in the model.
pL	The number of nonlinear parameters.
pA	The number of linear parameters.
ntsSeed	Reset number for random generator or Time series number.
N	Total number of data points.
t1	The starting point for the estimation.
tN	The end point for the estimation.
gamma	Exponential weighting factor, default 0.99.
lambdaArt	Nonlinear parameters used to create the artificial data.
alphaArt	Linear parameters used to create the artificial data.
lambda	Exponential parameters in autoregressive models.
alpha	Weights in autoregressive models.

Table 11: Information stored in the structure *Prob.PartSep*

Field	**Description**
pSepFunc	Number of partially separable functions.
index	Index for the partially separable function to compute, i.e. if $i = index$, compute $f_i(x)$. If $index = 0$, compute the sum of all, i.e. $f(x) = \sum_{i=1}^{M} f_i(x)$.

Table 12: Information stored in the structure *Prob.GLOBAL*

Field	Description
iterations	Number of iterations, default 50.
MaxEval	Number of function evaluations, default 500.
Integers	Set of integer variables.
epsilon	Global/local weight parameter, default 10^{-4}.
K	The Lipschitz constant. Not used.
tolerance	Error tolerance parameter. Not used.
C	Matrix with all rectangle centerpoints.
D	Vector with distances from centerpoint to the vertices.
L	Matrix with all rectangle side lengths in each dimension.
F	Vector with function values.
d	Row vector of all different distances, sorted.
d_min	Row vector of minimum function value for each distance.
Split	$Split(i,j)$ is the number of splits along dimension i of rectangle j.
T	$T(i)$ is the number of times rectangle i has been trisected.
G	Matrix with constraint values for each point.
ignoreidx	Rectangles to be ignored in the rectangle selection procedure.
I_L	$I\_L(i,j)$ is the lower bound for rectangle j in integer dimension $I(i)$.
I_U	$I\_U(i,j)$ is the upper bound for rectangle j in integer dimension $I(i)$.
feasible	Flag indicating if a feasible point has been found.
f_min	Best function value found at a feasible point.
s_0	s_0 is used as $s(0)$.
s	$s(j)$ is the sum of observed rates of change for constraint j.
t	$t(i)$ is the total number of splits along dimension i.

Table 13: Information stored in the structure *Prob.USER*

Field	Description
f	Name of m-file computing the objective function $f(x)$.
g	Name of m-file computing the gradient vector $g(x)$. If *Prob.USER.g* is empty then numerical derivatives will be used.
H	Name of m-file computing the Hessian matrix $H(x)$.
c	Name of m-file computing the vector of constraint functions $c(x)$.
dc	Name of m-file computing the matrix of constraint normals $\partial c(x)/dx$.
d2c	Name of m-file computing the 2nd part of 2nd derivative matrix of the Lagrangian function, $\sum_i \lambda_i \partial^2 c(x)/dx^2$.
r	Name of m-file computing the residual vector $r(x)$.
J	Name of m-file computing the Jacobian matrix $J(x)$.
d2r	Name of m-file computing the 2nd part of the Hessian for nonlinear least squares problem, i.e. $\sum_{i=1}^{m} r_i(x) \frac{\partial^2 r_i(x)}{\partial x_j \partial x_k}$.

Table 14: Information stored in the structure *Prob.optParam.LineSearch*

Field	**Description**
sigma	Line search accuracy; $0 < sigma < 1$. $sigma = 0.9$ inaccurate line search. $sigma = 0.1$ accurate line search, default 0.9.
rho	Determines the ρ line, default 0.01.
tau1	Determines how fast step grows in phase 1, default 9.
tau2	How near end point of $[a, b]$, default 0.1.
tau3	Choice in $[a, b]$ phase 2, default 0.5.
eps1	Minimal length for interval $[a, b]$, default 10^{-7}.
eps2	Minimal reduction, default 10^{-12}.
MaxIter	Maximum number of line search iterations.

Table 15: Information stored in the global Matlab structure *Result*.

Field	Description
Iter	Number of major iterations.
MinorIter	Number of minor iterations.
ExitFlag	0 if convergence to local min. Otherwise errors.
Inform	Information parameter, type of convergence.
f_k	Function value at optimum.
g_k	Gradient value at optimum.
H_k	Hessian value at optimum.
B_k	Quasi-Newton approximation of the Hessian at optimum.
x_0	Starting point.
f_0	Function value at start i.e. $f(x\_0)$.
x_k	Optimal point.
v_k	Lagrange multipliers.
r_k	Residual at optimum.
J_k	Jacobian matrix at optimum.
c_k	Value of constraints at optimum.
cJac	Constraint Jacobian at optimum.
xState	State of each variable, described in Table 16 .
bState	State of each linear constraint, described in Table 17.
cState	State of each general constraint.
optParam	Structure with special fields for optimization parameters, see Table 6.
Name	Problem name.
P	Problem number.
p_dx	Matrix where each column is a search direction.
alphaV	Matrix where row i stores the steplengths tried for the i:th iteration.
x_min	Lowest x-values in optimization. Used for plotting.
x_max	Highest x-values in optimization. Used for plotting.
F_X	F_X is a global matrix with rows: [iter_no f(x)].
GLOBAL	Structure with special fields for global optimization, see Table 18.
SepNLLS	General result variable with fields *z* and *Jz*. Used when running separable nonlinear least squares problems
Solver	Solver used.
SolverAlgorithm	Solver algorithm used.
CPUtime	CPU time used.
REALtime	Real time elapsed.
Nflops	Number of floating point operations.
probType	TOMLAB problem type.
solvType	TOMLAB solver type.
FuncEv	Number of function evaluations needed.
GradEv	Number of gradient evaluations needed.
ConstrEv	Number of constraint evaluations needed.
ResEv	Number of residual evaluations needed.
JacEv	Number of Jacobian evaluations needed.
Prob	Problem structure, see Table 5.
plotData	Structure with plotting parameters.

Table 16: The state variable *xState* for the variable.

Value	Description
0	A free variable.
1	Variable on lower bound.
2	Variable on upper bound.
3	Variable is fixed, lower bound is equal to upper bound.

Table 17: The state variable *bState* for each linear constraint.

Value	Description
0	Inactive constraint.
1	Linear constraint on lower bound.
2	Linear constraint on upper bound.
3	Linear equality constraint.

Table 18: Information stored in the structure *Result.GLOBAL*

Field	Description
C	Matrix with all rectangle centerpoints in original coordinates.
D	Vector with distances from centerpoint to the vertices.
L	Matrix with all rectangle side lengths in each dimension.
F	Vector with function values.
d	Row vector of all different distances, sorted.
d_min	Row vector of minimum function value for each distance.
Split	$Split(i,j)$ is the number of splits along dimension i of rectangle j.
T	$T(i)$ is the number of times rectangle i has been trisected.
G	Matrix with constraint values for each point.
ignoreidx	Rectangles to be ignored in the rectangle selection procedure.
I_L	$I\_L(i,j)$ is the lower bound for rectangle j in integer dimension $I(i)$.
I_U	$I\_U(i,j)$ is the upper bound for rectangle j in integer dimension $I(i)$.
feasible	Flag indicating if a feasible point has been found.
f_min	Best function value found at a feasible point.
s_0	s_0 is used as $s(0)$.
s	$s(j)$ is the sum of observed rates of change for constraint j.
t	$t(i)$ is the total number of splits along dimension i.

Table 19: Matlab Optimization toolbox routines with a TOMLAB interface.

Function	Type of problem solved
constr	Constrained minimization.
leastsq	Nonlinear least squares.
fmins	Unconstrained minimization using Nelder-Mead type simplex search method.
fminu	Unconstrained minimization using gradient search.
lp	Linear programming.
qp	Quadratic programming.