NLPLIB TB 1.0 - A MATLAB Toolbox for Nonlinear Optimization and Parameter Estimation
The NLPLIB Toolbox for Nonlinear Programming in MATLAB

Kenneth Holmström

Applied Optimization and Modeling Group (**TOM**) [1]

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

Research Report in MATHEMATICS / APPLIED MATHEMATICS
Technical Report IMa-TOM-1997-2

16 November 1997 (Revised March 30, 1998)

Abstract

The MATLAB toolbox **NLPLIB TB** is a collection of MATLAB m-files that solves nonlinear optimization and parameter estimation problems. Both **NLPLIB** solvers and other general-purpose solvers are possible to use. The toolbox is used in our undergraduate courses in Optimization Theory and Mathematical Programming for engineers and mathematicians.

Included in **NLPLIB TB** are routines for unconstrained and constrained optimization unconstrained and constrained nonlinear least squares and quadratic programming. New algorithms are implemented for the nonlinear least squares problem to approximate sums of exponentials to empirical data.

NLPLIB TB has menu programs and driver routines for the different types of nonlinear optimization problems. Also included is a general graphical user interface (GUI) which greatly simpifies the use of the toolbox.

[1] The **TOM** home page is **http://www.ima.mdh.se/tom**.

1 Introduction

The MATLAB toolbox **NLPLIB TB** (NonLinear Programming LIBrary Toolbox) is part of **TOMLAB** [15, 14]. **NLPLIB TB** is a set of MATLAB m-files, which solves nonlinear optimization problems and nonlinear parameter estimation problems in operations research and mathematical programming. The focus is on dense problems. The toolbox is running in MATLAB 5.1 and partly in MATLAB 4.2c and works on both PC (NT4.0, Win95, Windows 3.11) and UNIX systems (SUN, HP).

The optimization problem to be solved is either selected using a interactive menu program, or by direct call to a parameter driven driver routine. The problem is solved using either a **NLPLIB TB** solver, a solver in the MATLAB Optimization Toolbox [4] or using a MEX-file interface to call a FORTRAN or C optimization code.

NLPLIB TB has interactive menu programs for unconstrained and constrained optimization, unconstrained and constrained nonlinear least squares and quadratic programming.

NLPLIB TB includes a graphical user interface (GUI) [6] where all types of predefined problems can be solved. The user has total control of all parameters and variables when using the GUI.

TOMLAB MEX-file interfaces for both PC and UNIX has been developed for the commercial optimization code MINOS 5.5 [20]. In **NLPLIB TB** MINOS is used to solve nonlinear programs, but is also used to solve linear programs in **OPERA TB** [13]. **TOMLAB** MEX-file interfaces working on both PC and UNIX have also been developed for the commercial codes from the Systems Optimization Laboratory (SOL), Department of Operations Research, Stanford University, Carlifonia; NPSOL 5.02 [10], NPOPT 1.0-10 (updated version of NPSOL), NLSSOL 5.0-2, QPOPT 1.0-10, LSSOL 1.05 and LPOPT 1.0-10. The aim is to expand this list in the near future.

NLPLIB TB has many predefined test problems. It is easy to try to solve any of these problems using any of the solvers present. The user can easily expand the set of test problems with his own problems.

NLPLIB TB was designed with the aim to simplify the solution of practical optimization problems. After defining a new problem in the **NLPLIB TB** format it is then possible to try to solve the problem using any available solver or method.

For two-dimensional nonlinear unconstrained problems, the menu programs support graphical display of the selected optimization problem as a mesh or contour plot. Furthermore, the search directions together with marks of the trial step lengths are displayed on the contour plot. For higher-dimensional problems, the contour plot is displayed in a two-dimensional subspace. Plots showing the estimated convergence rate and the sequence of function values are included. The GUI has the same graphical utilities as the menu programs.

For nonlinear least squares problems, a routine to plot the data against the starting model and the fitted model is included. **NLPLIB TB** has been used for several years in optimization courses both at Mälardalen University and at Uppsala University. It is distributed free of charge for academic purposes.

2 Installation of the toolbox NLPLIB TB

If **NLPLIB TB** is installed as a standalone toolbox, the following routines from the **OPERA TB** toolbox must be included: *lpsimp1*,*lpsimp2*, *mPrint, printmat, xprint*, *xprinte* and *xprinti*.

2.1 Installation on PC systems

NLPLIB TB is normally installed as part of **TOMLAB**, with the subpath /tomlab/nlplib. On PC systems a normal choice of full path is

\matlab\tomlab\nlplib

or

\matlab\toolbox\tomlab\nlplib.

This path must be added to the MATLAB search path. Before starting a session running **NLPLIB TB**, call *nlplibInit*, which sets the number of columns and declares the global variables used. If the user has a screen with more than 80 columns, the variable `MAXCOLS` in *nlplibInit* should be changed to the correct number.

2.2 Installation on UNIX systems

NLPLIB TB is normally installed as part of **TOMLAB**, with the subpath /tomlab/nlplib. A possible full path is

/home/tomlab/nlplib

or

/home/matlab/toolbox/tomlab/nlplib.

This path must be added to the MATLAB search path. Before starting a session running **NLPLIB TB**, call *nlplibInit*, which sets the number of columns and declares the global variables used. If the user has a screen with more than 80 columns, the variable `MAXCOLS` in *nlplibInit* should be changed to the correct number.

3 Overview of NLPLIB TB

The development of **NLPLIB TB** started in 1989. MATLABroutines were developed to illustrate numerical optimization algorithms and were used in computer exercises in different optimization courses. The first version of several of the routines was the result of computer exercises solved by the students.

The main purpose of the development of **NLPLIB TB** is to develop tools to be used in teaching optimization algorithms. The principle is that all algorithms discussed in a course should be available as an easy-to-read computer implemented algorithm.

The current status of **NLPLIB TB** is:

- 39 700 lines of MATLAB code in the directories *nlplib* and *nlpdemo*.
- 234 files with algorithms, utilities and predefined problems.
- Menu programs and driver routines for
  - unconstrained optimization.
  - quadratic optimization.
  - constrained optimization
  - nonlinear least squares.
  - constrained nonlinear least squares.
- New algorithms for the nonlinear parameter estimation problem of fitting sums of exponential functions to empirical data.
- A graphical user interface (GUI) from which all types of problems can be solved (Only working in MATLAB 5.1).
- Most of the code is compatible with both MATLAB 4.2c and MATLAB 5.1.
- Suitable for teaching basic courses in optimization and mathematical programming;

3.1 Optimization Algorithms and Solvers

In Table 1 the optimization solvers in **NLPLIB TB** are shown. The solver for unconstrained optimization, *ucSolve*, and the solver routine for nonlinear least squares, *lsSolve*, are both written as a prototype routine. The prototype algorithm *lsSolve* for nonlinear least squares is described in detail in Section 3.2.

The routine *ucSolve* implements a prototype algorithm for **unconstrained optimization** with simple bounds on the parameters (**uc**), i.e. solves the problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & x_L \ \leq \ x \ \leq \ x_U \ , \end{aligned} \tag{1}$$

where $x, x_L, x_U \in \Re^n$ and $f(x) \in \Re$. *ucSolve* includes several of the most popular search step methods for unconstrained optimization. Bound constraints are treated as described in Gill et al.[11]. The search step methods for unconstrained optimization included in *ucSolve* are

- The Newton method.
- The quasi-Newton BFGS method (safeguarded).
- The quasi-Newton inverse BFGS method (safeguarded).
- The quasi-Newton DFP method (safeguarded).
- The quasi-Newton inverse DFP method (safeguarded).

- The Fletcher-Reeves conjugate gradient method.
- The Polak-Ribiere conjugate gradient method.
- 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 [17]. The code for the quasi-Newton matrix update is using safe guarding techniques to avoid rank problem in the updated matrix.

Another unconstrained solver in **NLPLIB TB** is the structural trust region algorithm *sTrustR*, combined with an initial trust region radius algorithm. The code is based on the algorithms in [5] and [21], but using the same framework as *ucSolve* and the same treatment of problems with bound constraints.

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

$$\begin{aligned} \min_x \quad & f(x) \\ \\ \text{subject to} \quad & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \end{array} , \end{aligned} \tag{2}$$

where $x, x_L, x_U \in \Re^n$, $f(x) \in \Re$, $A \in \Re^{m_1 \times n}$, $b_L, b_U \in \Re^{m_1}$ and $c_L, c(x), c_U \in \Re^{m_2}$. For general constrained nonlinear optimization a sequential quadratic programming (SQP) method by Schittkowski [22] is implemented in the routine *conSolve*.

A **Quadratic program** (**qp**) is defined as

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

where $c, x, x_L, x_U \in \Re^n$, $F \in \Re^{n \times n}$, $A \in \Re^{m_1 \times n}$, and $b_L, b_U \in \Re^{m_1}$. Quadratic programs are solved with a standard active set method [18], implemented in the routine *qpSolve*. The algorithm explicitly treats both inequality and equality constraints as well as lower and upper bounds on the variables (simple bounds). It converges for some indefinite **qp**s, but the code is not entirely robust for indefinite problems.

NLPLIB TB includes two algorithms for solving **qp** restricted to equality constraints only; a null space method (*qpe*) and Lagrange's method *qplm*).

3.2 Nonlinear Least Squares Algorithms

The **Nonlinear Least Squares Problem** (**ls**) is defined as

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2} r(x)^T r(x) \\ \\ \text{subject to} \quad & x_L \leq x \leq x_U , \end{aligned} \tag{4}$$

Table 1: Optimization solvers in **NLPLIB TB**.

Function	**Description**
ucSolve	A prototype routine for unconstrained optimization with simple bounds on the parameters. Implements Newton, quasi-Newton and conjugate gradient type of search methods.
lsSolve	A prototype algorithm for nonlinear least squares with simple bounds. Implements Gauss-Newton and hybrid quasi-Newton and Gauss-Newton type of methods.
clsSolve	A prototype algorithm for constrained nonlinear least squares. Currently handles simple bounds and linear equality and inequality constraints. Implements Gauss-Newton and hybrid quasi-Newton and Gauss-Newton type of search step methods.
conSolve	Constrained nonlinear minimization solver using sequential quadratic programming.
sTrustR	Solver for unconstrained optimization, using a structural trust region algorithm.
qpSolve	Solves a general quadratic program.
qpe	Solves a **qp** problem, restricted to equality constraints, using a null space method.
qplm	Solves a **qp** problem, restricted to equality constraints, using Lagrange's method.

where $x, x_L, x_U \in \Re^n$ and $r(x) \in \Re^N$. .

In **NLPLIB TB** the prototype nonlinear least squares algorithm *lsSolve* treats problems with bound constraints in a similar way as the routine *ucSolve* described in Section 3. In Figure 1 the algorithm is described in a flow sheet. There are six methods to compute the search direction. The two last, which are using inverses, are not practical methods, and included only to illustrate the drawbacks of using them.

If rank problems occur the prototype algorithm is using subspace minimization, see Lindström [17]. It is only possible to use this method and determine the pseudo rank using the first four methods to compute the search direction. The line search algorithm used is the same as for unconstrained problems.

The prototype routine *lsSolve* includes four optimization methods for nonlinear least squares problems:

- The Gauss-Newton method.
- The Al-Baali-Fletcher hybrid method [1].
- The Fletcher-Xu hybrid method [8].
- The Huschens TSSM method [16].

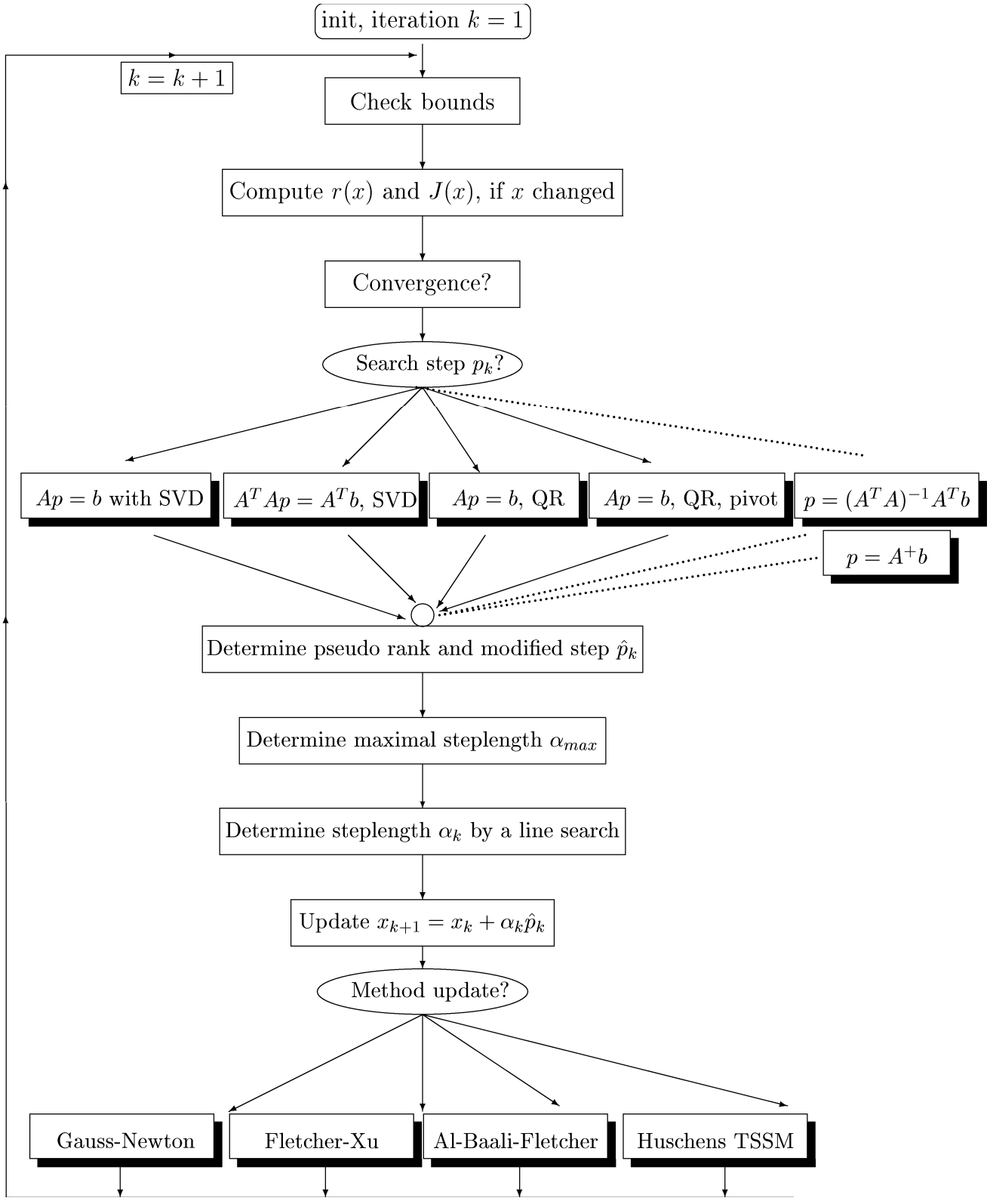

Figure 1: The **NLPLIB TB** prototype routine for nonlinear least squares

The **Constrained Nonlinear Least Squares Problem** (**cls**) is defined as

$$\begin{array}{lll} \min\limits_{x} & f(x) = \frac{1}{2} r(x)^T r(x) & \\ & & \\ & \begin{array}{rcccl} x_L & \leq & x & \leq & x_U \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \end{array} & , \end{array} \tag{5}$$

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

To handle constrained nonlinear least squares problems a new prototype routine *clsSolve* based on *lsSolve* is developed. Currently *clsSolve* can treat linear equality and inequality constraints, but the intention is to develop it further to handle general constraints. The search methods in *clsSolve* are the same as in *lsSolve*. Constraints are treated using an active set strategy.

4 The routines in NLPLIB TB

There are five menu programs defined in Table 2, one for each type of optimization problem. Included in the table is also the graphical user interface (GUI) for nonlinear programming, which has the same functionality in one routine as all the menu programs.

Table 2: Menu programs.

Function	**Description**
nlplib	Graphical user interface (GUI) for nonlinear optimization. It can handle all types of nonlinear optimization problems.
ucOpt	Menu for unconstrained optimization.
qpOpt	Menu for quadratic programming.
conOpt	Menu for constrained optimization.
lsOpt	Menu for nonlinear least squares problems
clsOpt	Menu for constrained nonlinear least squares problems

The menu programs described in Table 2 calls the corresponding driver routine given in Table 3.

For each external FORTRAN or C solver, a special driver routine is developed, which simplifies the use of the external solver.

Like the MATLAB Optimization Toolbox, **OPERA TB** and **NLPLIB TB** are using a vector with optimization parameters. In Optimization Toolbox, the routine *foptions* is setting up the default values in a vector OPTIONS with 18 parameters. Our solvers need more parameters, currently 40, and therefore the routine *goptions* is used instead of *foptions*. For each type of optimization problem there is a corresponding definition routine which calls goptions and defines default parameter values for the vector optPar. The routines are described in Table 4.

Table 3: Driver routines.

Function	**Description**
ucRun	Driver routine for unconstrained optimization.
qpRun	Driver routine for quadratic programs.
conRun	Driver routine for constrained optimization.
lsRun	Driver routine for nonlinear least squares problems.
clsRun	Driver routine for constrained nonlinear least squares problems.
minosRun	Driver and test routine for MINOS 5.5.
npsolRun	Driver and test routine for NPSOL 5.02.
npoptRun	Driver and test routine for NPOPT 1.0-10.
nlssolRun	Driver and test routine for NLSSOL 5.0-2.
qpoptRun	Driver and test routine for QPOPT 1.0-10.
lpoptRun	Driver and test routine for LPOPT 1.0-10.
lssolRun	Driver and test routine for LSSOL 1.05.

Table 4: Routines defining default optimization parameters.

Function	**Description**
ucDef	Define default parameters for unconstrained optimization.
qpDef	Define default parameters for quadratic programming.
conDef	Define default parameters for constrained optimization.
lsDef	Define default parameters for nonlinear least squares optimization.
clsDef	Define default parameters for constrained nonlinear least squares optimization.
goptions	Set default values for optimization and display parameters. Generalized version of *foptions* in Optimization Toolbox.

In Table 5 the utility functions needed by the solvers in Table 1 are displayed. The function *ittr* implements the initial trust region radius algorithm by Sartenaer [21].

The line search algorithm *linesrch* used by the solvers *conSolve*, *lsSolve*, *clsSolve* and *ucSolve* is a modified version of an algorithm by Fletcher Fletcher [9, chap. 2]. The use of quadratic (*intpol2*) and cubic interpolation (*intpol3*) is possible in the line search algorithm.

The routine *presolve* is running a presolve analysis on a system of linear equalities, linear inequalities and simple bounds. An algorithm by Gondzio [12] is implemented in *presolve*.

Table 5: Utility routines for the optimization solvers.

Function	**Description**
ittr	Initial trust region radius algorithm.
linesrch	Line search algorithm by Fletcher.
intpol2	Find the minimum of a quadratic interpolation. Used by *linesrch*.
intpol3	Find the minimum of a cubic interpolation. Used by *linesrch*.
presolve	Presolve analysis on linear constraints and simple bounds.

4.1 MEX-file interfaces

It is possible to call FORTRAN and C/C++ code from MATLAB using dynamically linked subroutines, so called MEX-file interfaces. On PC Windows systems these interfaces were previously either built using 16-bit DLL files or 32-bit REX MEX-file interface. MATLAB 5.1 now supports 32-bit DLLs. A severe restriction is that the standard I/O routines in C and FORTRAN are not available; DLL MEX-files must use the Windows file I/O functions instead. For long it has been very difficult to develop a working MEX-file interface on PC Windows systems, especially MEX-file interfaces to FORTRAN code. Most optimization solvers are written in FORTRAN and most of them also draws heavy on a functioning I/O system.

On UNIX systems it is easier to develop working MEX-file interfaces, and therefor FORTRAN MEX-file code is available for several optimization solvers.

In **NLPLIB TB** there is currently seven MEX-file interface developed, to the commercial solvers MINOS, NPSOL, NPOPT, NLSSOL, QPOPT, LPOPT and LSSOL as described in Table 6. As standard, MINOS has a very advanced I/O handling, but is also possible to switch it off and use MINOS as a silent subroutine. The other routines are also possible to make totally silent. The MEX-file interfaces are all written in C and compiled and linked using the WATCOM C/C++ version 10.6 compiler. It was impossible to make them work on PC systems without converting all FORTRAN code to C using *f2c* [7].

Table 6: MEX-file interface routines.

Function	**Description**
minos	Calls *minosMex*, the MEX-file interface routine for MINOS 5.5.
npsol	Calls *npsolMex*, the MEX-file interface routine for NPSOL 5.02.
npopt	Calls *npoptMex*, the MEX-file interface routine for NPOPT 1.0-10.
nlssol	Calls *nlssolMex*, the MEX-file interface routine for NLSSOL 5.0-2.
qpopt	Calls *qpoptMex*, the MEX-file interface routine for QPOPT 1.0-10.
lpopt	Calls *lpoptMex*, the MEX-file interface routine for LPOPT 1.0-10.
lssol	Calls *lssolMex*, the MEX-file interface routine for LSSOL 1.05.

4.2 Low level routines and Test Problems

We define the low level routines as the routines that compute the objective function value, the gradient vector, the Hessian matrix (second derivative matrix), the residual vector (for NLLS problems), the Jacobian matrix (for NLLS problems), the vector of constraint functions, the matrix of constraint normals and the second part of the second derivative of the Lagrangian function. The last three routines are only needed for constrained problems. The names of these routines are defined in a set of global variables with predefined names. It is the task of the problem setup routine, in **NLPLIB TB** routines with names of the type **_prob*, to return these values, and the driver routine then sets the global variables to correct values.

As input argument to all menu programs and driver routines the user can send his own problem setup routine. There is also a menu of all problem setup routines available for the current type of solver. This implies that the system is totally flexible. All information about a problem is stored in a structure variable *prob*. This variable is an argument to all low level routines and in the field element *prob.uP* all problem specific information needed to evaluate the low level routines is then easily retrieved. After writing the setup routine and the low level routines needed, **NLPLIB TB** is ready to run the users own problem.

Table 7 shows the reserved global variable names for the low level routines and the information that should be computed when running them. Only the routines relevant for a certain type of optimization problem need to be coded. There are dummy routines for the other routines.

Table 7: Global parameter names.

Global variable	Information computed running feval (global variable)
p_f	The objective function value $f(x)$.
p_g	The gradient vector $g(x)$ of the function $f(x)$.
p_H	The Hessian matrix (second derivative matrix) $H(x)$ of the function $f(x)$.
p_c	The vector of constraint functions $c(x)$.
p_dc	The matrix of constraint normals, $\partial c(x)/dx$.
p_d2c	The second part of the second derivative of the Lagrangian function, $\sum_i \lambda_i \partial^2 c_i(x)/dx^2$.
p_r	The residual vector $r(x)$ of the function $f(x) = \frac{1}{2} r(x)^T r(x)$.
p_J	The Jacobian matrix $J(x)$, where $J_{ij} = \partial r_i/dx_j, i = 1, ..., m, j = 1, ..., n$.

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, it was necessary to develop interface routines that returns the information in the format needed for the actual solver. The interface routines are using the MATLAB m-file name stored as a string in the global variables in Table 7 to call the correct low level routine. The Table 8 gives the current set of interface routines needed.

The table 9 describes the low level test functions and the corresponding problem setup routines needed for the predefined unconstrained optimization (**uc**) problems.

The Table 10 describes the low level test functions and the corresponding problem setup routines needed for the predefined constrained optimization (**con**) problems.

To be able to solve Quadratic Programming (**qp**) problems using general purpose solvers, routines to compute the objective function, the gradient vector and the Hessian matrix must be defined. The Table 11 describes the routine which defines the **qp** problem, *qpGetMat*, and the three other functions needed.

The Table 12 describes the low level routines, the initialization routine and the routines to compute the residual vector and the Jacobian matrix needed for the predefined nonlinear least squares (**ls**) test problems.

The problem of fitting positive 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. Some 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. The Table 13 describes the relevant routines:

The Table 14 describes the low level routines and the initialization routines needed for the predefined constrained nonlinear least squares (**cls**) test problems.

The Table 15 describes the low level test functions and the corresponding problem setup routines needed for the predefined unconstrained and constrained optimization problems from the CUTE data base [2, 3].

There are some options in the menu programs to display graphical information for the selected problem. For two-dimensional nonlinear unconstrained problems, the menu programs support graphical display of the selected optimization problem as mesh or contour plots. On the contour plot the iteration steps are displayed. For higher-dimensional problems, iterations steps are displayed in two-dimensional subspaces. The Table 16 describes the plot routines.

The Table 17 describes the utility routines called from other **NLPLIB TB** functions.

The optimization solvers in the MATLAB Optimization Toolbox which are possible to use in **NLPLIB TB** are listed in Table 18. The user must of course have a valid license. The driver routine checks if the actual routine is in the path, possible to call, and then uses **feval** to run it.

5 The User Interface and Menu Systems

This section describes the menu routines *ucOpt*, *qpOpt*, *conOpt*, *lsOpt* and *clsOpt*. It also discusses the functional structure of **TOMLAB** as displayed in Figure 5. The graphical user interface (GUI) has the same functionality as the menu programs. The GUI is presented in detail in [6]. The following is a list of the standard menu choices for unconstrained and constrained optimization:

- Name of the problem setup file and the problem to be solved.

- Should the problem be solved using default parameters or should problem dependent questions be asked?
- The amount of output and any restriction on the number of array elements displayed. This is to avoid too much output for large size problems.
- Optimization method (algorithm).
- Print levels and pause/no pause after each iteration.
- Optimization parameters of the following type:
  - σ, the line search accuracy.
  - The maximal number of iterations.
  - The starting values for the unknown variables x, and lower and upper bounds on x.
  - Choice if to use quadratic or cubic interpolation in line search algorithm.
  - A best guess of the lower bound on the optimal objective function value (used by the line search algorithm).
  - The tolerance on the convergence for the iterative sequence of the variables x, a convergence tolerance on the objective function value $f(x)$, on the norm of the gradient vector $g(x)$ and on the norm of the directed derivative $p^T g(x)$, $p = x_{k+1} - x_k$.
  - The maximal constraint violation for the inequality constraints $c(x) \geq 0$ and equality constraints $c(x) = 0$.
  - The rank test tolerance which determines the pseudo rank used in the subspace minimization technique. The subspace minimization technique is part of the determination of the search direction in some of the **NLPLIB** internal solvers.
- **Optimize**. Start an optimization with the selected optimization solver.
- Draw a contour plot of $f(x)$, and also draw the search directions p. Mark line search step length trials α_i for each search direction.
- Draw a mesh plot of $f(x)$.
- Draw other types of graphics, e.g. the objective function value for each iteration or the estimated linear convergence rate for each iteration.

Every parameter has initial default values. The user selects new values or simply uses the default values. When the user selects the option *Optimize*, the menu system calls the driver routine, the *Optimization Driver* box in Figure 5. The name of the routine which defines the optimization problem is one of the parameters given in the call to the driver. The driver routine calls this definition routine, the dashed *Setup Problem* box in Figure 5. All problem setup routines in **TOMLAB** have two different modes of behavior. If the problem number already is defined, the problem setup routine silently defines the problem. Otherwise, a menu is displayed, letting the user select the wanted problem. The

user may set the problem number in a direct call to the *Optimization Driver*, symbolized by the *Advanced User* box in Figure 5. This is useful if a large set of problems is to be solved, for example when trying out algorithms for a certain applied problem.

The *Setup Problem* routine defines a string matrix with the names of the m-files that computes the different elements defining the problem. These are the objective function value $f(x)$, the gradient vector $g(x)$, the Hessian matrix (matrix of second derivatives) $H(x)$, the vector of constraint functions $c(x)$, the matrix of constraint normals $dc(x)$ and the second part of the Hessian of the Lagrangian, $d2c(x)$. For nonlinear least squares problems the routines computing the residual vector $r(x)$ and the Jacobian matrix $J(x)$ are defined. The *Optimization Driver* defines global strings with the given function names, which are used in all computations.

The *Optimization Driver* either calls a **NLPLIB TB** or **OPERA TB** solver, a routine from MATLAB Optimization Toolbox or a routine callable using a predefined MEX-file interface.

The *Interface Routines* in Figure 5 are needed to convert computational results to the form needed by different solvers.

The names of the *Low Level Routines* in Figure 5 are the global strings defined in the *Optimization Driver*.

The solvers or the *Interface Routines* are running the MATLAB function *feval* on these global strings to compute the wanted functional quantities.

In Figure 2 an example is shown for the case of solving generally constrained nonlinear optimization problems. The user calls the menu program *conOpt*, defines the different options and start an optimization with the menu option *Optimize*, which makes *conOpt* call *conRun*. If the user has not chosen a name for the *Setup Problem* routine, *conRun* calls the Initialization routine *InitFile* using the default problem setup routine, *con_prob*. The *con_prob* setup routine is defining the name *con_f* for the MATLAB m-file that computes the objective function value. The names for the gradient vector, Hessian matrix, vector of constraint functions matrix of constraint normals and second part of the second derivative of the Lagrangian are con_g, *con_H*, *con_c con_dc* and *con_d2c*

These names are stored in the global strings, named *p_f*, *p_g*, *p_H*, *p_c*, *p_dc* and *p_d2c*. Then *conRun* calls the selected solver, either *conSolve* in **NLPLIB TB**, *constr* in MATLAB Optimization Toolbox or any of the three commercial solvers MINOS, NPSOL or NPOPT from Systems Optimization Laboratory (SOL).

There are several *Interface Routines* needed. 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*. MINOS, NPSOL and NPOPT instead need both the objective function value and the gradient vector to be returned in one call, which is the output of *nlp_fg*. The matrix of constraint normals should be supplied together with the vector of constraint functions. For NPSOL and NPOPT the routine *nlp_cdc* returns these both parameters. The matrix of constraint normals is stored sparse for MINOS, so the sparse *Interface Routine nlp_cdcS* is needed.

One of the menu options is to draw a contour plot of $f(x)$ together with the search steps. On each search step there are marks for each trial value the line search algorithm

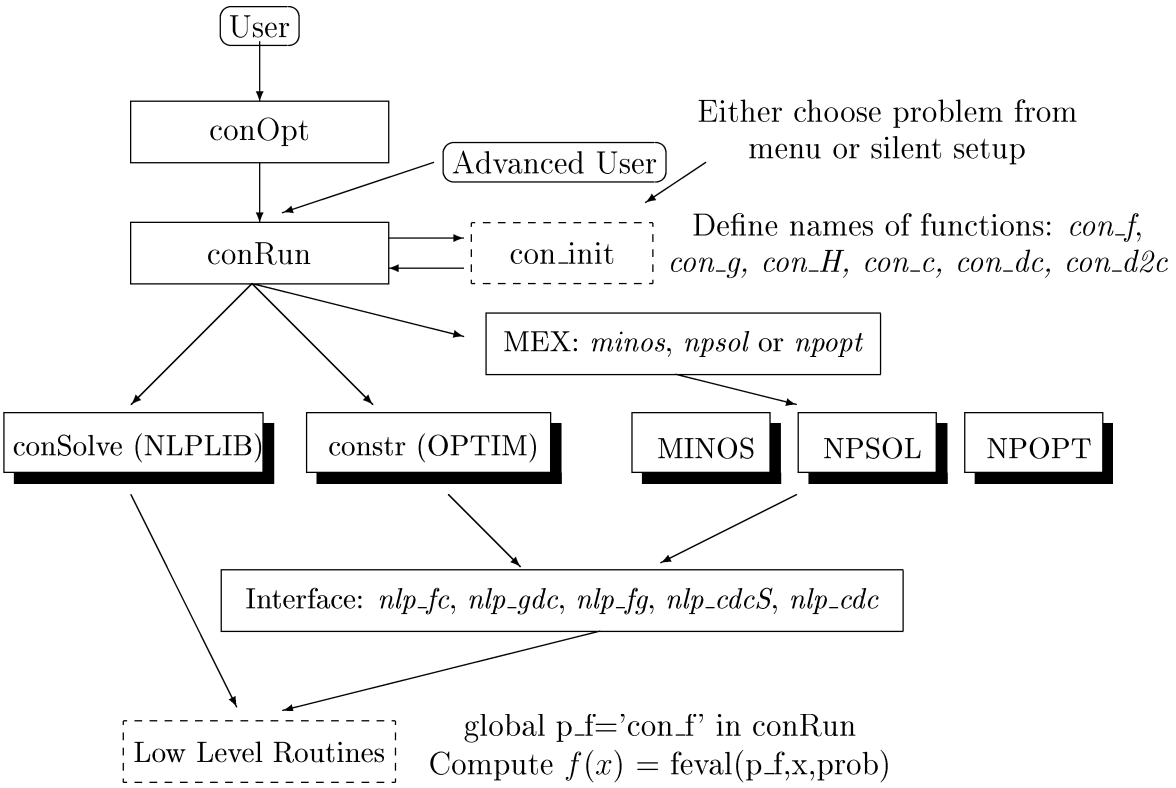

Figure 2: Solution of constrained nonlinear problems in **TOMLAB**.

for each trial value, where the line search algorithm had to evaluate the function. It is possible to follow the full iterative sequence on two-dimensional problems. We have run the prototype unconstrained solver *ucSolve* using two different methods. In Figure 3 the result of optimizing the classical Rosenbrock banana function, see [19] or [11, page 95], using Newtons method are displayed. There are a few steps where the line search has shortened the step. In contrast to this, see the behavior of the Fletcher-Reeves conjugate gradient method in Figure 4. This method (not using second derivative information) has a much more chaotic path to the solution. Such graphs can be illustrative for students in a first course in optimization.

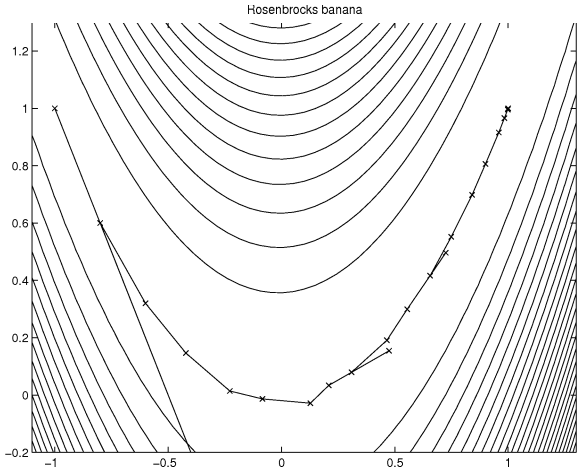

Figure 3: The Rosenbrock banana function with search directions and marks for the line search trials running *ucsolve* using the Newtons method.

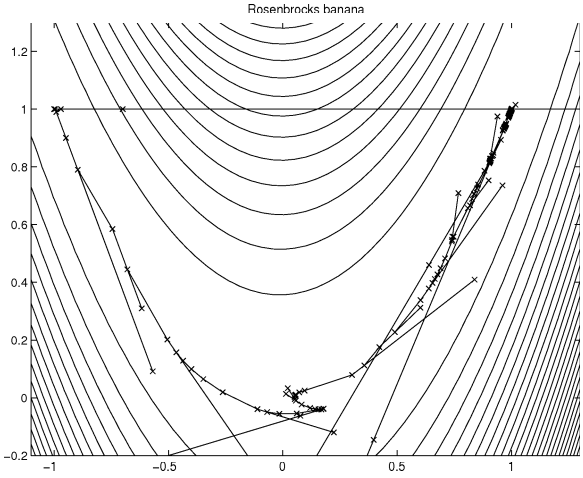

Figure 4: The Rosenbrock banana function with search directions and marks for the line search trials running *ucsolve* using the Fletcher-Reeves conjugate gradient method.

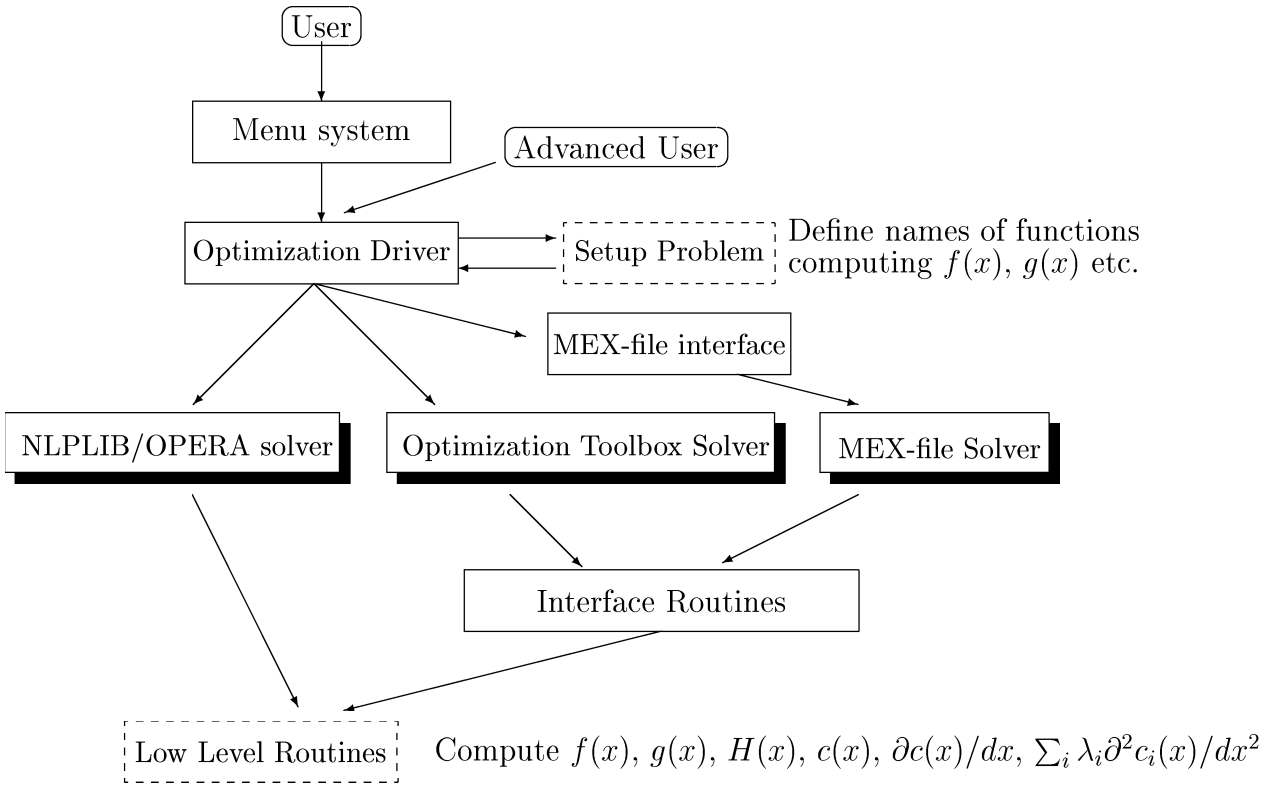

Figure 5: The process of optimization in **TOMLAB**.

6 Conclusions

NLPLIB TB is suitable for computer based learning of optimization and in computer exercises. Currently used at Mälardalen University and at Uppsala University. Now being introduced at Umeå University and Linköping University.

It may be used as a research tool to solve applied optimization problems. Currently used in our applied research in the Applied Optimization and Modeling Group.

NLPLIB TB is a flexible tool, with both a graphical user interface, menu programs and driver routines. It is powerful as an environment for solving optimization problems, with an increasing list of callable solvers.

References

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

[2] 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.

[3] 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.

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

[5] 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.

[6] Erik Dotzauer and Kenneth Holmström. A Graphical User Interface for Nonlinear Programming in MATLAB. *Annals of Operations Research*, Modeling Languages and Approaches:Submitted, 1998.

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

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

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

[10] 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.

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

[12] Jacek Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9(1):73–91, 1997.

[13] Kenneth Holmström. OPERA TB 1.0 - A MATLAB Toolbox for Optimization Algorithms in Operations Research. Technical Report IMa-TOM-1997-1, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997.

[14] Kenneth Holmström. TOMLAB - A General Purpose, Open MATLAB Environment for Research and Teaching in Optimization. Technical Report IMa-TOM-1997-3, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997. Presented at the 16th International Symposium on Mathematical Programming, Lausanne, Switzerland, August 24-29, 1997.

[15] Kenneth 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.

[16] 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.

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

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

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

[20] Bruce A. Murtagh and Michael 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.

[21] 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.

[22] 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.

Table 8: Interface routines for test problems.

Function	**Description**
nlp_fc	Compute function value $f(x)$ and the vector of constraint functions $c(x)$ using the MATLAB m-file names stored in the global variables *p_f* and *p_c*. This routine is used when calling routines from MATLAB Optimization Toolbox.
nlp_gdc	Compute the gradient $g(x)$ and the matrix of constraint normals $dc(x)$ using the MATLAB m-file names stored in the global variables *p_g* and *p_dc*. This routine is used when calling routines from MATLAB Optimization Toolbox.
nlp_fg	Compute function value $f(x)$ and gradient vector $g(x)$ using the MATLAB m-file names stored in the global variables *p_f* and *p_g*. This routine is used when calling MINOS.
nlp_cdc	Compute constraint vector $c(x)$ and matrix of constraint normals $dc(x)$ using the MATLAB m-file names stored in the global variables *p_c* and *p_dc*.
nlp_cdcS	Compute constraint vector $c(x)$ and matrix of constraint normals $dc(x)$ (stored sparse) using the MATLAB m-file names stored in the global variables stored in the global variables *p_c* and *p_dc*. This routine is used when calling MINOS.
nlp_c	Generate empty constraint for an unconstrained problem, for use with constrained codes (dummy routine).
nlp_dc	Generate derivative of empty constraint (dummy routine).
nlp_cD	Generate dummy constraint, for use with constrained codes that need at least one constraint defined.
nlp_dcD	Generate derivative of dummy constraint defined in nlp_cD.
nlp_JT	Compute transposed Jacobian matrix using the MATLAB m-file name stored in the global variable *p_J*. This routine is used when calling routines from MATLAB Optimization Toolbox.
funobj	Dummy problem for solvers accessed using the MEX-file interface. funobj computes the function value $f(x)$ and the gradient vector $g(x)$.
funcon	Dummy problem for solvers accessed using the MEX-file interface. funcon computes the vector of constraint functions $c(x)$ and the matrix of constraint normals $dc(x)$.
ls_rJ	Computes the residual vector $r(x)$ and the Jacobian matrix $J(x)$ using the MATLAB m-file names stored in the global strings *p_r* and *p_J*. This routine is used when calling NLSSOL.

Table 9: Unconstrained nonlinear (**uc**) test problems.

Function	**Description**
uc_prob	Initialization of **uc** test problems.
uc_f	Compute the objective function $f(x)$ for **uc** problems.
uc_g	Compute the gradient vector $g(x)$ for **uc** problems.
uc_H	Compute the Hessian matrix $H(x)$ for **uc** problems.

Table 10: Generally constrained nonlinear (**con**) test problems.

Function	**Description**
con_prob	Initialization of **con** test problems.
con_f	Compute the objective function $f(x)$ for **con** test problems.
con_g	Compute the gradient $g(x)$ for **con** test problems.
con_H	Compute the Hessian matrix $H(x)$ of $f(x)$ for **con** test problems.
con_c	Compute the constraint residuals $c(x)$ for **con** test problems.
con_dc	Compute the derivative of the constraint residuals for **con** test problems.
con_fm	Compute merit function $\theta(x_k)$.
con_gm	Compute gradient of merit function $\theta(x_k)$.

Table 11: Quadratic programming (**qp**) test problems.

Function	**Description**
qp_prob	Initialization of **qp** test problems.
qpGetMat	Return the matrices and vectors defining a **qp**.
qp_f	Compute the function value $f(x) = \frac{1}{2}x^T Fx + c^T x$ for **qp** test problems.
qp_g	Compute the gradient vector $g(x) = Fx + c$ for **qp** test problems.
qp_H	Compute the constant Hessian $H(x) = F$ for **qp** test problems.

Table 12: Nonlinear least squares (**ls**) test problems.

Function	**Description**
ls_prob	Initialization of **ls** test problems.
ls_r	Compute the residual vector $r_i(x), i = 1, ..., m$. $x = (x_1, ..., x_n)^T$ for **ls** test problems.
ls_J	Compute the Jacobian matrix $J_{ij}(x) = \partial r_i / dx_j, i = 1, ..., m, j = 1, ..., n$ for **ls** test problems.
ls_f	General routine to compute the objective function value $f(x) = \frac{1}{2} r(x)^T r(x)$ for nonlinear least squares type of problems. Using global variable *p_r* to evaluate $r(x)$.
ls_g	General routine to compute the gradient $g(x) = J(x)^T r(x)$ for nonlinear least squares type of problems. Using global variable *p_J* to evaluate $J(x)$.
ls_H	General routine to compute the Hessian approximation $H(x) = J(x)^T * J(x)$ for nonlinear least squares type of problems.

Table 13: Exponential fitting test problems.

Function	**Description**
exp_artP	Generate artificial exponential sum problems
exp_init	Setup routine for both ordinary nonlinear least squares and separable nonlinear least squares formulation.
exp_prob	Defines a exponential fitting type of problem, with data series (t, y). The file includes data from several different empirical test series.
exp_r	Compute the residual vector $r_i(x), i = 1, ..., m$. $x \in \Re^n$
exp_J	Compute the Jacobian matrix $\partial r_i / dx_j, i = 1, ..., m, j = 1, ..., n$.
exp_H	Compute approximate second derivatives. Adding extra positive definite term $Q(x)$, which gives $H(x) = J(x)^T * J(x) + Q(x)$.
exp_c	Compute the constraints $\lambda_1 < \lambda_2 < ...$ on the exponential parameters $\lambda_i, i = 1, ..., p$.
exp_dc	Compute matrix of constraint normals for constrained exponential fitting problem.
exp_d2c	Compute second part of second derivative matrix of the Lagrangian function for constrained exponential fitting problem. This is a zero matrix, because the constraints are linear.
exp_q	Find starting values for exponential parameters $\lambda_i, i = 1, ..., p$.
exp_p	Find optimal number of exponential terms, p.
exp_geo	Utility routine used by *exp_q*.
exp_eq	Utility routine used by *exp_q*.
exp_root	Utility routine used by *exp_q*.

Table 14: Constrained nonlinear least squares (**cls**) test problems.

Function	**Description**
cls_prob	Initialization of **cls** test problems.
cls_r	Compute the residual vector $r_i(x), i = 1, ..., m.\ x \in \Re^n$ for **cls** test problems.
cls_J	Compute the Jacobian matrix $J_{ij}(x) = \partial r_i / dx_j, i = 1, ..., m, j = 1, ..., n$ for **cls** test problems.
cls_c	Compute the vector of constraint functions $c(x)$ for for **cls** test problems.
cls_dc	Compute the matrix of constraint normals $dc(x)$ for for **cls** test problems.
cls_d2c	Compute the second part of the second derivative of the Lagrangian function for **cls** test problems.

Table 15: Test problems from CUTE data base.

Function	**Description**
ctools	Interface routine to constrained CUTE test problems.
utools	Interface routine to unconstrained CUTE test problems.
cto_prob	Initialization of constrained CUTE test problems.
ctl_prob	Initialization of large constrained CUTE test problems.
cto_f	Compute the objective function $f(x)$ for constrained CUTE test problems.
cto_g	Compute the gradient $g(x)$ for constrained CUTE test problems.
cto_H	Compute the Hessian $H(x)$ of $f(x)$ for constrained CUTE test problems.
cto_c	Compute the vector of constraint functions $c(x)$ for constrained CUTE test problems.
cto_dc	Compute the matrix of constraint normals for constrained CUTE test problems.
uto_prob	Initialization of unconstrained CUTE test problems.
utl_prob	Initialization of large unconstrained CUTE test problems.
uto_f	Compute the objective function $f(x)$ for unconstrained CUTE test problems.
uto_g	Compute the gradient $g(x)$ for unconstrained CUTE test problems.
uto_H	Compute the Hessian $H(x)$ of $f(x)$ for unconstrained CUTE test problems.

Table 16: Plot routines.

Function	**Description**
plotiter	Setup call to plot routines *step2d* and *step2dx*.
step2d	Draws a contour plot of the objective function $f(x)$ and all search steps for two dimensional optimization problems. The line search steps at each step are marked.
step2dx	Draws a contour plot of the objective function $f(x)$ and plot the search direction two dimensional subspace when the optimization problem has more than two unknown parameters.
meshf	Draws a mesh plot of the function $f(x)$.
lineplot	Plot along a line (e.g. view the line search problem).

Table 17: Utility routines.

Function	**Description**
OparMenu	Subprogram displaying menus which gives the user possibility to change optimization parameters.
opt1,opt2,opt3	Part of the code for the menu programs: conOpt, lsOpt, ucOpt, clsOpt, ucOpt.
run1,run2,run3	Part of the code for the driver routines: conRun, lsRun, ucRun, clsRun, ucRun.
sol1,sol2	Part of the code for the NPSOL and NPOPT solvers: npsolRun, npoptRun.
secUpdat	Compute least change formulation of secant updates, called from *lsSolve*.
inputSet	Prompt user for an integer value from a set of values.
inputR	Prompt user for a real number in a given interval.
inputV	Prompt user for n-dimensional vector where all elements belong to a given interval. If the user gives only one value, the whole vector is set to this value. vector
circinit	Generate random points for the nonlinear least squares Circle Fitting Problem.
circplot	Plot actual and approximated circle and simulated data points for the nonlinear least squares Circle Fitting Problem.
backsub	Solves upper triangular linear system with backward substitution
inirobot	Extra routine needed for the robot problem, constrained test problem number 12.

Table 18: Optimization toolbox routines.

Function	**Type of Problem Solved**
constr	Constrained minimization.
leastsq	Nonlinear least squares.
fminu	Unconstrained minimization using gradient search.
lp	Linear programming.
qp	Quadratic programming.
nnls	Nonnegative linear least squares (no license needed).
conls	Constrained linear least squares.