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

The TOMLAB NLPLIB Toolbox for Nonlinear Programming [1]

Kenneth Holmström[2] and Mattias Björkman[3]
Center for Mathematical Modeling,
Department of Mathematics and Physics
Mälardalen University, P.O. Box 883, SE-721 23 Västerås, Sweden

Abstract

The paper presents the toolbox NLPLIB TB 1.0 (NonLinear Programming LIBrary); a set of Matlab solvers, test problems, graphical and computational utilities for unconstrained and constrained optimization, quadratic programming, unconstrained and constrained nonlinear least squares, box-bounded global optimization, global mixed-integer nonlinear programming, and exponential sum model fitting.

NLPLIB TB, like the toolbox OPERA TB for linear and discrete optimization, is a part of TOMLAB; an environment in Matlab for research and teaching in optimization. TOMLAB currently solves small and medium size dense problems.

Presently, NLPLIB TB implements more than 25 solver algorithms, and it is possible to call solvers in the Matlab Optimization Toolbox. MEX-file interfaces are prepared for seven Fortran and C solvers, and others are easily added using the same type of interface routines. Currently, MEX-file interfaces have been developed for *MINOS*, *NPSOL*, *NPOPT*, *NLSSOL*, *LPOPT*, *QPOPT* and *LSSOL*. There are four ways to solve a problem: by a direct call to the solver routine or a call to a multi-solver driver routine, or interactively, using the Graphical User Interface (GUI) or a menu system. The GUI may also be used as a preprocessor to generate Matlab code for stand-alone runs. If analytical derivatives are not available, automatic differentiation is easy using an interface to ADMAT/ADMIT TB. Furthermore, five types of numerical differentiation methods are included in NLPLIB TB.

NLPLIB TB implements a large set of standard test problems. 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 and NLPLIB TB have been used to solve several applied optimization problems. New types of algorithms are implemented for the nonlinear least squares problem to approximate sums of exponential functions to empirical data and for global optimization. We present some preliminary test results, which show very good performance for the NLPLIB TB solvers.

Keywords: Nonlinear Programming, MATLAB, CUTE, AMPL, Graphical User Interface, Software Engineering, Mathematical Software, Optimization, Algorithms, Exponential Sum Fitting, Nonlinear Least Squares.

AMS Subject Classification: 90C30, 90C20, 90C45

---

[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**.
[3] E-mail: **mbk@mdh.se**.

1 Introduction

The toolbox NLPLIB TB (NonLinear Programming LIBrary Toolbox) is part of TOMLAB ; an environment in Matlab for research and teaching in optimization [19, 18, 17, 16]. 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.x and works on both PC (NT4.0, Windows 95/98, Windows 3.11) and UNIX systems (SUN, HP).

Currently NLPLIB TB consists of about 54000 lines of m-file code (in 300 files) implementing algorithms, utilities and predefined problems, all well documented in the User's Guide [22]. The User's Guide includes descriptions and examples of how to define and solve optimization problems, as well as detailed descriptions of the routines.

The optimization problem to be solved is either selected using a interactive menu program, or directly defined in a call to a multi-solver driver routine. The problem is solved using either a NLPLIB TB solver, a solver in the Matlab Optimization Toolbox [5] 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, quadratic programming, box-bounded global optimization and global mixed-integer nonlinear programming.

NLPLIB TB includes a graphical user interface (GUI) [8] where all types of predefined problems can be solved. Using the GUI the user has total control of all optimization parameters and variables.

TOMLAB MEX-file interfaces for both PC and UNIX has been developed for the commercial optimization code MINOS 5.5 [30]. In TOMLAB, MINOS is used to solve nonlinear programs in NLPLIB TB, and linear programs in OPERA TB [21]. 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, California; NPSOL 5.02 [13], 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 implements a large set of 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. 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 options 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. Also included are new algorithms for the nonlinear parameter estimation problem of fitting sums of exponential functions to empirical data.

In Section 2 the different optimization algorithms and solvers in NLPLIB TB are discussed. Some other important utility routines are discussed in Section 3, e.g. different types of differentiation. Some information about the MEX-file interfaces, the low-level routines and the test problems are given in Section 3.1. The Section 4 discusses the most frequently menu and plot options used. In Section 5 we present three areas, where TOMLAB and NLPLIB TB have been a valuable tool: constrained nonlinear least squares; the special case of exponential sum fitting problems and box-bounded global optimization. Some test results are presented for these application areas, showing good performance for the NLPLIB TB solvers. We end by some conclusions in Section 6.

2 Optimization Algorithms and Solvers

In this section we discuss the optimization problems that NLPLIB TB are able to solve. In Table 1 the optimization solvers in NLPLIB TB are listed. The solver for unconstrained optimization, *ucSolve*, and the nonlinear least squares solvers *lsSolve* and *clsSolve*, are all written as prototype routines, i.e. the routines implements several optimization algorithms in one code. This simplifies maintenance and further algorithm development.

Table 1: Optimization solvers in NLPLIB TB.

Function	**Description**
ucSolve	A prototype routine for unconstrained optimization with simple bounds on the variables. Implements Newton, four quasi-Newton and three conjugate-gradient methods.
glbSolve	A routine for box-bounded global optimization.
gblSolve	Stand-alone version of *glbSolve*. Runs independently of NLPLIB TB.
glcSolve	A routine for global mixed-integer nonlinear programming.
gclSolve	Stand-alone version of *glcSolve*. Runs independently of NLPLIB TB.
lsSolve	A prototype algorithm for nonlinear least squares with simple bounds. Implements Gauss-Newton, and hybrid quasi-Newton and Gauss-Newton methods.
clsSolve	A prototype algorithm for constrained nonlinear least squares. Currently handles simple bounds and linear equality and inequality constraints using an active set strategy. Implements Gauss-Newton, and hybrid quasi-Newton and Gauss-Newton methods.
conSolve	Constrained nonlinear minimization solver using two different sequential quadratic programming methods.
nlpSolve	Constrained nonlinear minimization solver using filter SQP.
sTrustR	Solver for constrained convex optimization of partially separable functions, using a structural trust region algorithm.
qpBiggs	Solves a quadratic program.
qpSolve	Solves a quadratic program.
qpe	Solves a quadratic program, restricted to equality constraints, using a null space method.
qplm	Solves a quadratic program, restricted to equality constraints, using Lagrange's method.

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

$$\begin{array}{ll} \min\limits_{x} & f(x) \\ & \\ \text{s.t.} & x_L \leq x \leq x_U, \end{array} \tag{1}$$

where $x, x_L, x_U \in \mathbb{R}^n$ and $f(x) \in \mathbb{R}$. *ucSolve* includes several of the most popular search step methods for unconstrained optimization. Bound constraints are treated as described in Gill et al. [14]. The search step methods for unconstrained optimization included in *ucSolve* 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 [27]. The quasi-Newton codes also use safe guarding techniques to avoid rank problem in the updated matrix.

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

$$
\begin{array}{ll}
\min\limits_{x} & f(x) \\
& \\
\text{s.t.} & \begin{array}{rcccl} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \\ c_L & \leq & c(x) & \leq & c_U \end{array}
\end{array}
\tag{2}
$$

where $x, x_L, x_U \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$. For general constrained nonlinear optimization a sequential quadratic programming (SQP) method by Schittkowski [34] is the main method implemented in the routine *conSolve*. *conSolve* also includes the Han-Powell SQP method. A third SQP type algorithm is the Filter SQP by Fletcher and Leyffer [11], implemented in *nlpSolve*.

Another constrained solver in NLPLIB TB is *sTrustR*, implementing a structural trust region algorithm combined with an initial trust region radius algorithm. The code is based on the algorithms in [6] and [33], and treats partially separable functions. If not using an analytical Hessian, safeguarded BFGS or DFP are used for the Quasi-Newton update. Currently, *sTrustR* only solves problems where the feasible region defined by the constraints is convex.

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

$$
\begin{array}{ll}
\min\limits_{x} & f(x) = \frac{1}{2} x^T F x + c^T x \\
& \\
\text{s.t.} & \begin{array}{rcccl} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array}
\end{array}
\tag{3}
$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. Quadratic programs are solved with a standard active-set method [28], implemented in the routine *qpSolve*. *qpSolve* explicitly treats both inequality and equality constraints, as well as lower and upper bounds on the variables (simple bounds). For indefinite quadratic programs, it is using directions of negative curvature in the process of finding a local minimum.

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

The **nonlinear least squares problem (ls)** is defined as

$$
\begin{array}{ll}
\min\limits_{x} & f(x) = \frac{1}{2} r(x)^T r(x) \\
& \\
\text{s.t.} & x_L \leq x \leq x_U,
\end{array}
\tag{4}
$$

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

In NLPLIB TB the prototype nonlinear least squares algorithm *lsSolve* treats problems with bound constraints in a similar way as the routine *ucSolve*.

The prototype routine *lsSolve* includes four optimization methods for nonlinear least squares problems: the Gauss-Newton method, the Al-Baali-Fletcher [1] and the Fletcher-Xu [12] hybrid method, and the Huschens TSSM method [23]. If rank problems occur, the prototype algorithm is using subspace minimization. The line search algorithm used is the same as for unconstrained problems.

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

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

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

The 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 routine *glbSolve* implements an algorithm for **box-bounded global optimization** (**glb**), i.e. problems of the form (1) that have finite simple bounds on all the variables. *glbSolve* implements the DIRECT algorithm [25], which is a modification of the standard Lipschitzian approach that eliminates the need to specify a Lipschitz constant. In *glbSolve* no derivative information is used. For **global mixed-integer nonlinear programming** (**glc**), *glcSolve* implements an extended version of DIRECT (Jones [24]), that handles problems with both nonlinear and integer constraints.

For global optimization problems with expensive function evaluations the routine *ego* implements the Efficient Global Optimization (EGO) algorithm [26]. The idea of the EGO algorithm is to first fit a response surface to data collected by evaluating the objective function at a few points. Then, EGO balances between finding the minimum of the surface and improving the approximation by sampling where the prediction error may be high.

3 Other Routines in NLPLIB TB

There are seven menu programs defined in Table 2, one for each type of optimization problem (*probType*). 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. Handles all types of nonlinear optimization problems.
ucOpt	Menu for unconstrained optimization.
glbOpt	Menu for box-bounded global optimization.
glcOpt	Menu for global mixed-integer nonlinear programming.
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 with the same *probType*, any of *ucRun*, *glbRun*, *glcRun*, *qpRun*, *conRun*, *lsRun* or *clsRun*.

In Table 3 the utility functions needed by the solvers in Table 1 are displayed. The function *ittr* implements the initial trust region radius algorithm by Sartenaer [33].

The line search algorithm *LineSearch* used by the solvers *conSolve*, *lsSolve*, *clsSolve* and *ucSolve* is a modified version of an algorithm by Fletcher [10, chap. 2]. The use of quadratic (*intpol2*) and cubic interpolation (*intpol3*) is possible in the line search algorithm. For more details, see [22].

The routine *preSolve* is running a presolve analysis on a system of linear equalities, linear inequalities and simple bounds. An algorithm by Gondzio [15] is implemented in *preSolve*.

Instead of analytical derivatives, it is easy to use either any of five types of numerical differentiation, or automatic differentiation using an interface to the toolbox ADMAT TB. For information of how to obtain a copy of ADMAT TB, see the URL: **http://simon.cs.cornell.edu/home/verma/AD/**.

The default numerical differentiation type is an implementation of the FD algorithm [14, page 343]. It is the classical approach with forward or backward differences, together with an automatic step selection procedure. If the Spline Toolbox is installed, gradient, Jacobian, constraint gradient and Hessian approximations could be computed in three different ways using either of the routines *csapi*,

Table 3: Utility routines for the optimization solvers.

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

csaps or *spaps*. Numerical differentiation is automatically used for gradient, Jacobian, constraint gradient and Hessian if the user routine is not present. First order derivatives could also be estimated by use of complex variables. This approach avoids the subtractive cancellation error inherent in the classical derivative approximation, see [35].

3.1 MEX-file Interfaces

In NLPLIB TB there is currently seven MEX-file interfaces developed, to the commercial solvers MINOS, NPSOL, NPOPT, NLSSOL, QPOPT, LPOPT and LSSOL. As standard, MINOS has a very advanced input-output 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, after converting the Fortran code to C using *f2c* [9].

3.2 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. Only the routines relevant for a certain type of optimization problem need to be coded. There are dummy routines for the other routines. If routines that are computing derivatives are undefined (function name variables are set as empty), NLPLIB TB automatically uses numerical differentiation.

All information about a problem is stored in a structure variable *Prob*, described in detail in [19] and the User's Guide [22]. This structure variable is an argument to all low level routines. If the user needs to supply information to the low level routines, this information should be put in the vector field element *Prob.uP*, or as arbitrarily sub fields to *Prob.USER*. By this way, information needed to evaluate the low level routines is easily retrieved. Normally the user also writes a setup routine for the initialization process of problems, together with the low level routines. It is also possible to avoid this setup routine, and directly solve problems, only defining the low level routines (the NLPLIB TB QuickRun option).

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.

Table 4 describes the low level test functions and the corresponding problem setup routines needed for the predefined constrained optimization (**con**) problems. For the predefined unconstrained optimization (**uc**) problems and the quadratic programming problems (**qp**) similar routines are needed.

The problem of fitting positive sums of positively weighted exponential functions to empirical data

Table 4: 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 $\theta(x_k)$.

may be formulated either as a nonlinear least squares problem or a separable nonlinear least squares problem. Several empirical data series are predefined and artificial data series may also be generated. Algorithms to find starting values for different number of exponential terms are implemented. Table 5 describes the relevant routines.

Table 5: Exponential fitting test problems.

Function	Description
exp_ArtP	Generate artificial exponential sum problems.
expInit	Find starting values for the exponential parameters λ.
exp_prob	Defines a exponential fitting type of problem, with data series (t, y). The file includes data from several different empirical test series.
Helax_prob	Defines 335 medical research problems supplied by Helax AB, Uppsala, where an exponential model is fitted to data. The actual data series (t, y) are stored on one file each, i.e. 335 data files, 8MB large, and are not distributed. A sample of five similar files are part of *exp_prob*.
exp_r	Compute the residual vector $r_i(x), i = 1, ..., m$. $x \in \mathbb{R}^n$
exp_J	Compute the Jacobian matrix $\partial r_i / dx_j, i = 1, ..., m, j = 1, ..., n$.
exp_d2r	Compute the 2nd part of the second derivative for the nonlinear least squares exponential fitting problem.
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 model fitting problem.
exp_d2c	Compute second part of second derivative matrix of the Lagrangian function for constrained exponential model 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.

Table 6 describes the low level routines and the initialization routines needed for the predefined constrained nonlinear least squares (**cls**) test problems. Similar routines are needed for the nonlinear least squares (**ls**) test problems (no constraint routines needed).

Table 7 describes the low level test functions and the corresponding problem setup routines needed for the unconstrained and constrained optimization problems from the CUTE data base [4, 3].

There are several 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

Table 6: Constrained nonlinear least squares (**cls**) test problems.

Function	**Description**
cls_prob	Initialization of **cls** test problems.
cls_r	Compute the residual vector $r_i(x), i = 1, ..., m$. $x \in \mathbb{R}^n$ 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 **cls** test problems.
cls_dc	Compute the matrix of constraint normals $\partial c(x)/dx$ for for **cls** test problems.
cls_d2c	Compute the second part of the second derivative of the Lagrangian function for **cls** 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.
ls_g	General routine to compute the gradient $g(x) = J(x)^T r(x)$ for nonlinear least squares type of problems.
ls_H	General routine to compute the Hessian approximation $H(x) = J(x)^T * J(x)$ for nonlinear least squares type of problems.

displayed in two-dimensional subspaces. Special plots for nonlinear least squares problems, plotting model against data, are also available, as well as plots of line search problem, plots of circles approximating points in the plane for the Circle Fitting Problem etc.

4 The Menu Systems

This section gives a brief description of options when running the menu routines *ucOpt*, *ucOpt*, *qpOpt*, *conOpt*, *lsOpt* and *clsOpt*. The Graphical User Interface (GUI) has the same functionality as the menu programs. The GUI is presented in detail in [8]. The following is a list of most of the menu choices:

- 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?
- Optimization algorithm and solver.
- Optimization solver sub-method.
- Set optimization parameters of the following type:
  - Choice if to use automatic differentiation.
  - Method how to approximate derivatives.
  - The line search accuracy σ.
  - The maximal number of iterations.
  - Starting values and lower and upper bounds for the unknown variables 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 $p^T g(x)$, $p = x_{k+1} - x_k$.

Table 7: 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.
cto_d2c	Compute the second part of the second derivative of the Lagrangian function 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.

  - The maximal violation for the bounds, and the linear and nonlinear constraints.
  - 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 TB internal solvers.
  - Choice if to use a separable nonlinear least squares algorithm or not.

- Print levels and pause/no pause after each iteration.
- **Optimize**. Start an optimization with the selected optimization solver.
- Re-Optimize with the latest solution as starting value. Useful to determine if the optimal point is really found.
- Plot options:
  - 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 $f(x)$.
  - Plot data against the starting model and the fitted model for parameter estimation problems.
  - 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.

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 where the line search algorithm had to evaluate the objective 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 1 the result of optimizing the classical Rosenbrock banana function, see [29] or [14, 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

2. 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.

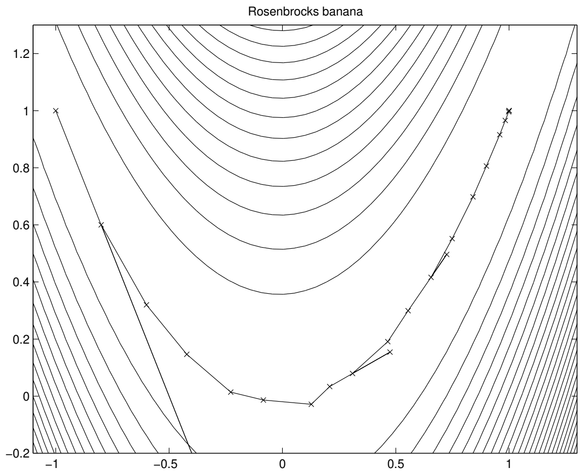

Figure 1: The Rosenbrock banana function with search directions and marks for the line search trials running *ucsolve* using the Newtons method.

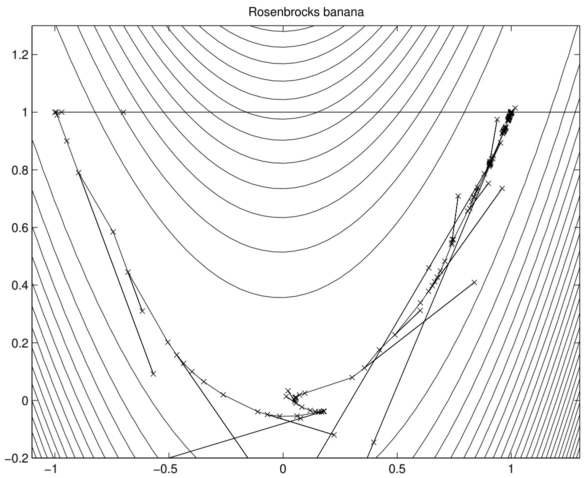

Figure 2: The Rosenbrock banana function with search directions and marks for the line search trials running *ucsolve* using the Fletcher-Reeves conjugate-gradient method.

5 Examples and Applications

In this section we will present results showing that the NLPLIB TB routines perform well on standard test problems as well as on real life applications. For nonlinear least squares problems and exponential sum model fitting problems comparisons between the NLPLIB TB solvers and commercial solvers are made.

5.1 Nonlinear Least Squares Problems

The NLPLIB TB nonlinear least squares solvers *lsSolve* and *clsSolve* are used in several of our research projects, e.g. estimation of formation constants in chemical equilibrium analysis, analysis of plasmid stability in fermentation processes and fitting of exponential sums to empirical data in radiotherapy planning. Examples of some exponential sum model fitting problems will be given in Section 5.2.

Here, we will present results from a comparison of our routines to *NLSSOL*, *NPOPT* and the Optimization Toolbox 1.5 routine *leastsq*. By our routines we mean *clsSolve* with the ordinary Gauss-Newton method (GN) and *clsSolve* with the Fletcher-Xu hybrid method (FX). A standard test set for nonlinear squares problems is the test set of More, Garbow and Hillstrom [29]. The solvers are applied to the 35 problems of the test set. The results are presented in Table 8. Each entry consists of three integers, which give the number of iterations, residual evaluations and Jacobian evaluations required to solve the problem. A $*$ indicates that the maximum number of allowed iterations (here 150) is reached and a bar indicates that the optimal function value has not been reached at all. The routine *leastsq* take as input the maximum number of allowed function evaluations instead of iterations so this limit will be reached before 150 iterations have been performed so the average values could be a bit misleading. This test is promising, because it shows that our routines performs even better than the commercial solvers, but we must keep in mind that this is not a full evaluation.

For a comparison of *clsSolve* and other solvers on linearly constrained nonlinear least squares problems, see the thesis of Björkman [2].

5.2 Exponential Sum Fitting Problems

In [20] algorithms for fitting exponential sums $D(r) = \sum_{i=1}^{p} a_i(1 - \exp(-b_i r))$ to numerical data are presented and compared for efficiency and robustness. The numerical examples stem from parameter estimation in dose calculation for radiotherapy planning. The doses are simulated by emitting an ionizing photon beam into water and at different depths and different radius from the beam center measuring the absorption. The absorbed dose is normally distinguished into primary dose from particles excited by the photon beam and scattered dose from the following particle interactions.

In Table 9 we present results from a comparison of the same type as presented in Section 5.1 for the Helax problems described above. The table entries consists of three integers which give the numbers of iterations, residual evaluations and Jacobian evaluations required to solve the problem. We restrict to present detailed information for the first fifteen and the last two problems but the average values and the number of failures are based on all the 334 problems. The † indicates that the separable nonlinear least squares algorithm II in [32] is run. The efficiency of the *clsSolve* Fletcher-Xu method with the separable algorithm is obvious, less than 65 percent of the number of iterations, residual evaluations and Jacobian evaluations for the best of the other solvers are required to solve the problems.

Worth mentioning is that NLPLIB TB includes very well performing routines for computing starting values for exponential sum model fitting problems, see the thesis by Petersson [31]. This is extremely important when solving problems in real life applications, and these good initial values are used in the test above.

5.3 Global Optimization Problems

As mentioned in Section 2, NLPLIB TB also includes routines for solving global optimization problems. The box-bounded constrained version and the general mixed integer constrained version of the DIRECT algorithm are implemented in *glbSolve* and *glcSolve* respectively. In Table 10 we show how *glbSolve* perform on the seven standard test functions from Dixon and Szegö [7] and two test functions from Yao [36]. The table entries give the number of function evaluations needed for convergence. Here, convergence is defined in terms of percent error from the known optimal function value. Let f_{global} denote the known optimal function value and let f_{min} denote the best

Table 8: Comparison of algorithmic performance for the More, Garbow, Hillstrom test set for nonlinear least squares problems.

	clsSolve GN	clsSolve FX	leastsq LM	leastsq GN	NLSSOL	NPOPT
MGH 01	18 33 33	19 36 36	18 20 20	15 19 19	32 57 57	23 27 27
MGH 02	—	—	—	—	—	7 9 9
MGH 03	15 17 17	15 17 17	84 100 100	62 77 77	—	—
MGH 04	13 42 42	14 32 32	—	15 26 26	15 24 24	—
MGH 05	6 8 8	6 8 8	13 14 14	7 8 8	12 14 14	16 20 20
MGH 06	—	15 48 48	14 15 15	21 41 41	22 83 83	24 52 52
MGH 07	—	—	—	—	—	—
MGH 08	5 6 6	5 6 6	15 16 16	6 7 7	10 13 13	17 22 22
MGH 09	1 2 2	1 2 2	3 4 4	4 5 5	2 2 2	5 9 9
MGH 10	8 12 12	8 12 12	—	46 47 47	—	—
MGH 11	—	—	—	—	—	—
MGH 12	5 6 6	5 6 6	22 23 23	9 11 11	9 11 11	28 34 34
MGH 13	10 11 11	10 11 11	41 42 42	15 16 16	27 27 27	62 68 68
MGH 14	50 97 97	44 57 57	74 82 82	46 58 58	47 78 78	39 49 49
MGH 15	10 14 14	6 10 10	21 22 22	22 24 24	13 28 28	20 33 33
MGH 16	150 419 419 *	16 27 27	35 42 42	119 151 151	150 319 319 *	18 24 24
MGH 17	8 11 11	9 16 16	29 46 46	10 11 11	37 68 68	45 70 70
MGH 18	—	19 67 67	72 82 82	73 85 85	—	—
MGH 19	11 16 16	12 19 19	23 25 25	149 151 151	19 31 31	60 86 86
MGH 20	4 5 5	4 5 5	113 151 151	112 151 151	—	82 91 91
MGH 21	18 33 33	18 34 34	21 22 22	15 19 19	32 56 56	62 107 107
MGH 22	10 11 11	10 11 11	42 43 43	15 16 16	28 28 28	97 111 111
MGH 23	55 59 59	19 32 32	31 37 37	143 151 151	141 377 377	145 205 205
MGH 24	145 319 319	150 262 262 *	144 145 145	96 109 109	—	—
MGH 25	9 10 10	9 10 10	7 8 8	3 4 4	14 14 14	18 19 19
MGH 26	7 14 14	7 14 14	9 10 10	8 11 11	10 21 21	—
MGH 27	13 51 51	9 33 33	12 13 13	29 47 47	12 23 23	15 17 17
MGH 28	2 3 3	2 3 3	5 6 6	4 5 5	4 4 4	18 32 32
MGH 29	2 3 3	2 3 3	5 6 6	4 5 5	4 4 4	6 6 6
MGH 30	4 5 5	4 5 5	8 9 9	5 6 6	6 6 6	21 38 38
MGH 31	5 6 6	5 6 6	7 8 8	6 7 7	8 8 8	—
MGH 32	1 2 2	1 2 2	1 10 10	1 2 2	1 1 1	1 3 3
MGH 33	1 2 2	1 2 2	2 12 12	2 12 12	3 6 6	2 3 3
MGH 34	1 2 2	1 2 2	2 12 12	2 12 12	3 6 6	2 3 3
MGH 35	—	—	—	—	—	—
Average	20 42 42	14 26 26	30 35 35	34 42 42	25 50 50	33 46 46
Failures	6	4	6	4	9	10

Table 9: Comparison of algorithmic performance for the Helax exponential sum fitting problems. The † indicates the use of a separable nonlinear least squares algorithm.

	clsSolve GN	clsSolve FX †	leastsq LM	leastsq GN	NLSSOL	NPOPT
Helax XP001	9 12 12	6 7 7	17 20 20	11 23 23	14 19 19	56 79 79
Helax XP002	8 11 11	6 7 7	17 26 26	10 13 13	9 13 13	53 77 77
Helax XP004	8 10 10	9 10 10	16 26 26	10 23 23	11 15 15	38 57 57
Helax XP005	8 10 10	10 11 11	20 46 46	10 13 13	11 15 15	37 56 56
Helax XP007	9 11 11	10 11 11	15 17 17	10 22 22	13 16 16	30 49 49
Helax XP008	9 11 11	12 14 14	15 17 17	10 13 13	13 16 16	28 44 44
Helax XP010	10 12 12	10 11 11	15 16 16	10 13 13	13 16 16	31 47 47
Helax XP011	10 12 12	11 12 12	16 17 17	9 11 11	13 20 20	20 25 25
Helax XP013	10 11 11	11 12 12	21 49 49	9 11 11	13 20 20	13 22 22
Helax XP014	15 21 21	20 30 30	27 28 28	14 23 23	15 23 23	34 48 48
Helax XP016	17 24 24	18 37 37	30 32 32	15 18 18	17 29 29	33 52 52
Helax XP017	19 25 25	19 26 26	33 35 35	14 16 16	25 36 36	33 71 71
Helax XP019	21 30 30	20 41 41	38 40 40	19 21 21	25 36 36	39 61 61
Helax XP020	23 31 31	21 34 34	44 46 46	21 25 25	25 36 36	41 63 63
Helax XP022	27 39 39	19 35 35	53 55 55	23 30 30	29 44 44	43 63 63
....	....	....	....	....	....	....
....	....	....	....	....	....	....
....	....	....	....	....	....	....
Helax XP499	9 10 10	5 6 6	13 14 14	9 10 10	7 9 9	17 25 25
Helax XP500	9 10 10	5 6 6	13 14 14	9 10 10	7 9 9	17 25 25
Average	16 18 18	8 10 10	29 34 34	13 16 16	13 18 18	21 31 31
Failures	0	0	0	0	0	0

function value at some point in the search, then the percent error is defined by

$$E = 100 \cdot \frac{f_{min} - f_{global}}{|f_{global}|}.$$

Table 10: Number of function evaluations needed by *glbSolve* for convergence on the Dixon-Szegö and Yao test functions.

Test function	function evaluations, $E < 1\%$	function evaluations, $E < 0.01\%$
Shekel 5	100	153
Shekel 7	94	143
Shekel 10	94	143
Hartman 3	70	178
Hartman 6	198	529
Branin	65	165
Goldstein-Price	83	167
Six-hump camel	77	146
Shubert	3193	3274

The test function *Shekel's foxholes* from the First International Contest on Evolutionary Optimization (ICEO) is in two dimensions illustrative to show how effective *glbSolve* could be on problems with several local (nonglobal) minima. A contour plot of the function with dots indicating points where the function value has been computed is shown in Figure 3.

After 21 iterations and 147 function evaluations, the best function value found by *glbSolve* is -12.119 at $x = (8.0239, 9.1467)^T$. As you can see in Figure 3 most of the sampled points are concentrated

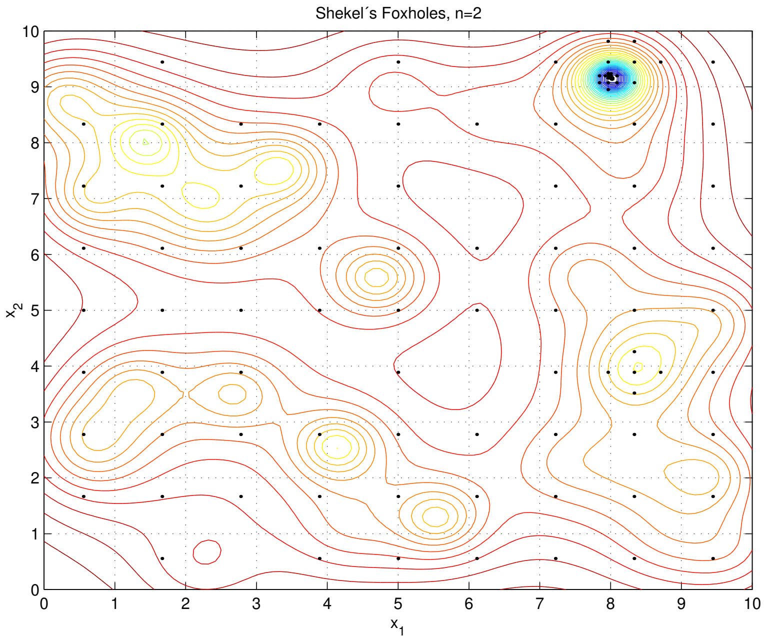

Figure 3: Contour plot of the Shekels's Foxholes function with 147 sampled points, using *glbSolve*.

in the area around the global minimum up in the right corner of the bounding box and very few function evaluations are wasted on sampling around the many local minima.

6 Conclusions

NLPLIB TB is a powerful tool for applied optimization research and algorithmic development in nonlinear programming. NLPLIB TB is also suitable for computer based learning of optimization and in computer exercises.

NLPLIB TB together with TOMLAB is a flexible tool, with both a graphical user interface, menu programs and multi-solver driver routines. The possibility to very easily use both automatic and numerical differentiation makes TOMLAB especially useful in practical applications, where derivative information may be hard to obtain. The global optimization routines are very suitable for the common case of parameter estimation in simulation models. The robust constrained nonlinear least squares routines are efficient tools in the solution of applied parameter estimation problems. With the open design and the interfaces the Fortran solvers, CUTE and AMPL, there are many possible ways to utilize the system.

Acknowledgements

We would like to thank Prof. Philip E. Gill for giving us access to the commercial codes NPSOL, NPOPT, NLSSOL, LPOPT, QPOPT and LSSOL. Also we are grateful to Prof. Michael Saunders for sending us the MINOS distribution. Prof. David M. Gay helped us making the AMPL interface work. We would also thank Prof. Yangquan Chen for a lot of help in making the CUTE interface work on PC systems. 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.

Thanks to Dr. Donald R. Jones who has, with the DIRECT and EGO algorithms, inspired us to include routines for global optimization problems. He has furthermore been very helpful in discussions on details of the algorithms.

The graduate students in the Applied Optimization Group (TOM) have made important contributions; Jöran Petersson has made important contributions to the routines for the exponential fitting problem as part of the exponential fitting project; Erik Dotzauer developed the first version of the solver *nlpSolve*. Many students have also contributed with bits and pieces, which we hereby acknowledge.

References

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

[2] M. Björkman, *Nonlinear Least Squares with Inequality Constraints*, Bachelor Thesis, Department of Mathematics and Physics, Mälardalen University, Sweden, 1998.

[3] 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.

[4] 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.

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

[6] 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.

[7] L. C. W. Dixon and G. P. Szegö, *The global optimisation problem: An introduction*, in Toward Global Optimization, L. Dixon and G. Szego, eds., New York, 1978, North-Holland Publishing Company, pp. 1–15.

[8] E. Dotzauer and K. Holmström, *The TOMLAB Graphical User Interface for Nonlinear Programming*, Advanced Modeling and Optimization, 1 (1999).

[9] 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.

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

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

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

[13] 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.

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

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

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

[17] ———, *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.

[18] ———, *TOMLAB - An Optimization Development Environment in Matlab*, Annals of Operations Research, Modeling Languages and Approaches (1998). Submitted.

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

[20] K. Holmström, A. Ahnesjö, and J. Petersson, *Algorithms for exponential sum fitting in radiotherapy planning*, (1999). To be submitted.

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

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

[23] 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.

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

[25] 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.

[26] 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.

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

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

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

[30] 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.

[31] 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.

[32] A. Ruhe and P.-Å. Wedin, *Algorithms for Separable Nonlinear Least Squares Problems*, SIAM Review, 22 (1980), pp. 318–337.

[33] 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.

[34] 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.

[35] W. SQUIRE AND G. TRAPP, *Using complex variables to estimate derivatives of real functions*, SIAM Review, 40 (1998), pp. 110–112.

[36] Y. YAO, *Dynamic tunneling algorithm for global optimization*, IEEE Transactions on Systems, Man, and Cybernetics, 19 (1989), pp. 1222–1230.