TOMLAB - A General Purpose, Open MATLAB Environment for Research and Teaching in Optimization [1]

Kenneth Holmström

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

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-1997-3

10 October 1997 (Revised March 30, 1998)

[1]Presented at the 16th International Symposium on Mathematical Programming, August 24-29, 1997, Lausanne, Switzerland.

Abstract

TOMLAB is a general purpose, open and integrated MATLAB environment for research and teaching in optimization on **UNIX** and **PC** systems. The motivation for **TOMLAB** is to simplify research on practical optimization problems, giving easy access to all types of solvers; at the same time having full access to the power of MATLAB.

By using a simple, but general input format, combined with the ability in MATLAB to evaluate string expressions, it is possible to run internal **TOMLAB** solvers, MATLAB Optimization Toolbox and commercial solvers written in FORTRAN or C/C++ using MEX-file interfaces. Currently MEX-file interfaces have been developed for *MINOS*, *NPSOL*, *NPOPT*, *NLSSOL*, *LPOPT*, *QPOPT* and *LSSOL*.

TOMLAB may either be used totally parameter driven or menu driven. The basic principles will be discussed. The menu system makes it suitable for teaching. Many standard test problems are included. More test problems are easily added. There are many example and demonstration files. Iteration steps including line search may be graphically displayed together with contour plots when running nonlinear optimization.

TOMLAB is based on **NLPLIB TB 1.0**, a MATLAB toolbox for nonlinear programming and parameter estimation and **OPERA TB 1.0**, a MATLAB toolbox for operations research, with emphasis on linear and discrete optimization. Over 50 different algorithms are implemented. Of special interest are the algorithms for general and separable nonlinear least squares. Our new implementation of the Fletcher-Xu hybrid method, the Al-Baali-Fletcher hybrid method and Huschens totally structured secant method (TSSM) give fast and robust convergence on ill-conditioned parameter estimation problems.

TOMLAB is free for academic purposes. Contribution from others are welcome, like more solvers, interfaces to other software packages, utilities and MEX-file interfaces.

More information on **TOMLAB** is found on **http://www.ima.mdh.se/tom**.

KEYWORDS: MATLAB, Optimization, Mathematical Software, Algorithms, Nonlinear Least Squares.

1 Introduction

This paper presents **TOMLAB**, an environment in MATLAB for the solution of optimization problems. **TOMLAB** features menu systems and driver routines for many common types of optimization problems. **TOMLAB** has many algorithms implemented in the toolboxes **NLPLIB TB 1.0** [29] and **OPERA TB 1.0** [30], but may also call routines from the MATLAB Optimization Toolbox [10]. Furthermore, problems may be solved running optimization solvers in FORTRAN and C/C++. This is possible using MEX-file interfaces.

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

In recent years several modeling languages has been developed, like AIMMS [7], AMPL [21], ASCEND [40], GAMS [8, 11] and LINGO [2]. The modeling system acts as a preprocessor. The user describes his problem in detail in a very verbal language, an opposite to a concise mathematical description of the problem. This problem description file is normally modified in a text editor, with help from example files solving the same type of problem. Much effort is directed to the development of more user friendly interfaces. The model system processes the input description file and calls any of the available solvers. For a solver to be available to the model system, a special type of interface has previously been written.

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

For people with a mathematical background, modeling languages often seems to be a very tedious way to define an optimization problem. There has been general attempts to find languages more suitable than FORTRAN or C/C++ to describe mathematical problems, like the compact and powerful APL language [33, 41]. Using APL, the author around 1985 very easy built an advanced interactive menu and graphical analysis system. The system was used by the research department at the Swedish National Industrial Board (SIND) and researchers in regional economy to analyze the Swedish industry using mathematical programming models [1].

Now, languages like MATLAB has a very rapid growth of users. MATLAB was originally created [36] as a preprocessor to the standard FORTRAN subroutine libraries in numerical linear algebra, LINPACK [13] and EISPACK [44] [22], very much the same idea as the modeling languages discussed above. MATLAB of today is a much more advanced and powerful tool, with graphics, animation and advanced menu design possibilities integrated with the mathematics. The MATLAB language has made the development of toolboxes possible, which serves as a direct extension to the language itself. Using MATLAB as an environment for solving optimization problems offers much more possibilities for analysis than just the pure solution of the problem.

Comparing the power of the MATLAB environment with that of the modeling systems I find it hard for the modeling systems to compete. The idea of this paper, and the concept of **TOMLAB** is to try to integrate all different systems, getting access to the best of all worlds. **TOMLAB** should be seen as a complement to existing model languages, for the user needing more power and flexibility than given by a modeling system.

This paper is organized as follows. In Section 2 we discuss the main motivations for **TOMLAB** and the concept. The next section, Section 3, describes the feature and structure of **TOMLAB**. We then describe the two most important parts of **TOMLAB**, the toolbox **OPERA TB 1.0** in Section 4 and the toolbox **NLPLIB TB 1.0** in Section 5. In Section 6 the nonlinear least squares solvers in **NLPLIB TB 1.0** are discussed. Finally, we end with some conclusions and a discussion about further work in Section 8.

2 The concept of TOMLAB

The starting point for **TOMLAB** was to develop a system that could be of use in education. The MATLAB language is very suitable for describing numerical algorithms and is similar to the type of algorithmic pseudo code often used to present algorithms. When the actual implementation is similar to the black board presentation, the students can easily follow the steps and get computational experience with the algorithm. The principle has been that all algorithms discussed in a course should be available as an easy-to-read computer implemented algorithm. The response has up-to-now been positive from the students using **TOMLAB**.

Another important motivation for the development of **TOMLAB** was to get a research environment for our group, the Applied Optimization and Modeling group. The applications we work with are often large, nonlinear and numerically ill conditioned. It is in most cases not possible to directly use standard programs. Therefore we need a combination of new algorithms and standard software. To find the best combination we need a flexible environment which allows us to perform tests on many different problem formulations and try out different solvers. Working in MATLAB has been a great benefit when developing algorithms to find the unknown speciation and the parameters in inorganic chemical equilibria [28, 32]. **TOMLAB** has also been of great use in our energy optimization project [14, 15] and in the development of algorithms for nonlinear parameter estimation [39].

We think that **TOMLAB** could be used to collect some single MATLAB research codes that otherwise would not be used. It is difficult to get an algorithm into a production code. Not all good algorithms find their way into such a code. At least it often takes long time.

Looking at the demands on **TOMLAB**, it must be easy for students to use. Therefore menu programs are needed. It should be easy to include new test problems of simple type. To be of use for applied research it must be easy to integrate the problem definitions independent of the language it has been written in. Commonly, problems are defined in MATLAB, but may also be in the form of FORTRAN or C/C++ routines. The principle must be that the problem should only needs to be defined once and then solved by any type of solver. It must also be easy to integrate new solvers, normally using a MEX-file interface.

The main problem with MEX-file interfaces is the weak support from the vendor of MATLAB, the Math Works, Inc., for FORTRAN on PC systems. There are already many ready-to-use MEX-file interfaces written in FORTRAN, which works well on UNIX systems, but are difficult to use on PC systems. Hopefully the support will improve. On PC we now have to use the freely available FORTRAN to C conversion routine *f2c* [17] to make the interfaces work.

3 The features and structure of TOMLAB

The main features of **TOMLAB** may be summarized as follows:

- **TOMLAB** is a MATLAB based environment.

- It implements >50 optimization algorithms in the toolbox **OPERA TB 1.0** and in the toolbox **NLPLIB TB 1.0**.
- It solves linear and discrete optimization problems like
  - linear programs,
  - network programs, with special treatment of transportation programs,
  - integer programs and
  - dynamic programming problems

  using the toolbox **OPERA TB 1.0**.
- It solves
  - unconstrained and constrained nonlinear optimization problems,
  - quadratic programs,
  - unconstrained and constrained nonlinear least squares problems and
  - fitting of positive sums of exponential functions to data

  using the toolbox **NLPLIB TB 1.0**.
- **TOMLAB** is portable, runs in MATLAB 5.1 and partly MATLAB 4.2c on
  - UNIX (SUN, HP) and
  - PC (NT4.0, Win95, Windows 3.11).
- Menu programs and driver routines makes **TOMLAB** very easy to use.
- **TOMLAB** is using MEX-file interfaces to run standard optimization software. The interfaces are available both for PC and UNIX. Currently MEX-file interfaces has been developed for the commercial codes MINOS, NPSOL, NPOPT, NLSSOL, QPOPT, LSSOL and LPOPT.
- It is possible to use many of the routines in MATLAB Optimization Toolbox.
- You only need to define your problem once and use all available solvers!

The functional structure of **TOMLAB** is displayed in Figure 1. A normal user runs the menu system for the actual problem type. 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.

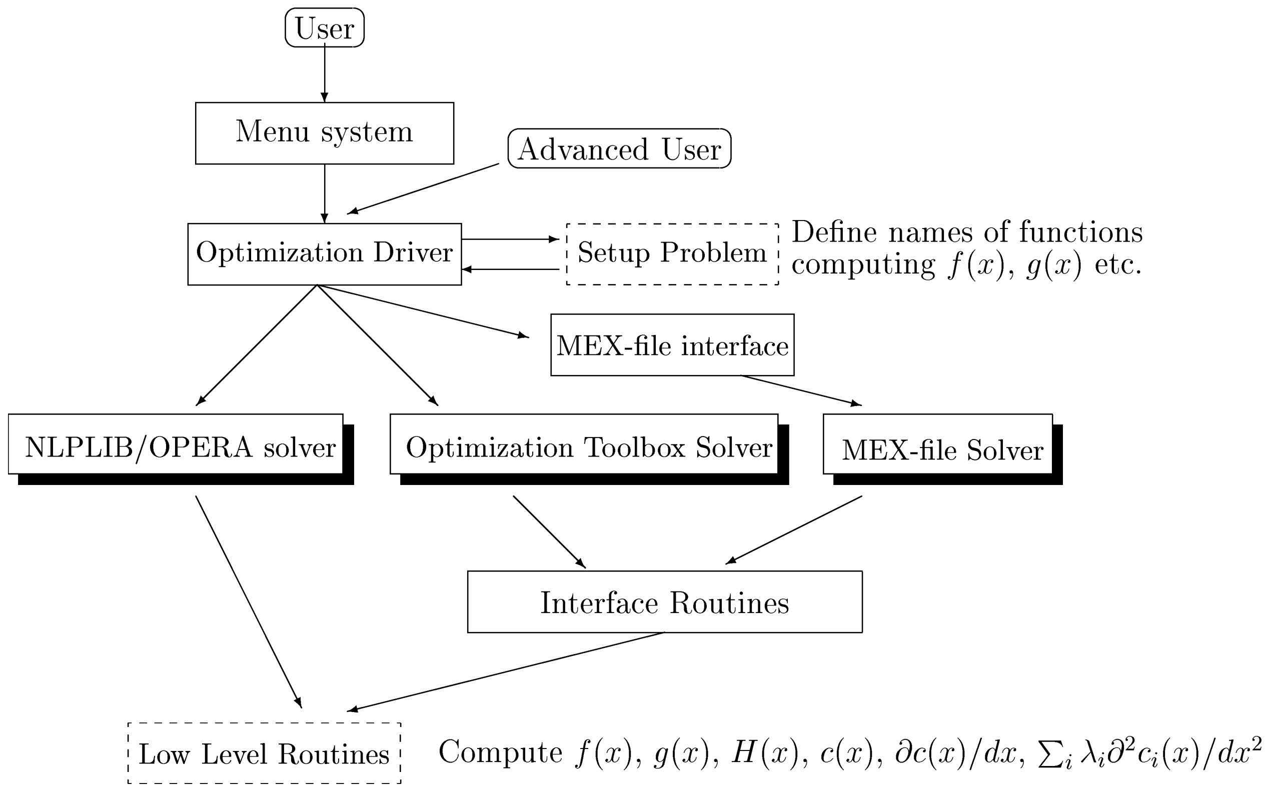
User
Menu system
Advanced User
Optimization Driver
Setup Problem
Define names of functions computing $f(x)$, $g(x)$ etc.
MEX-file interface
NLPLIB/OPERA solver
Optimization Toolbox Solver
MEX-file Solver
Interface Routines
Low Level Routines
Compute $f(x)$, $g(x)$, $H(x)$, $c(x)$, $\partial c(x)/dx$, $\sum_i \lambda_i \partial^2 c_i(x)/dx^2$

Figure 1: The process of optimization in **TOMLAB**.

- 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 TB** 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 1. 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 1. 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 1. 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 1.0** or **OPERA TB 1.0** solver, a routine from MATLAB Optimization Toolbox or a routine callable using a predefined MEX-file interface.

The *Interface Routines* in Figure 1 are needed to convert computational results to the form needed by different solvers.

The names of the *Low Level Routines* in Figure 1 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 1.0**, *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 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 [37] or [24, page 95], using Newtons method are displayed. There are a few steps where the line search has

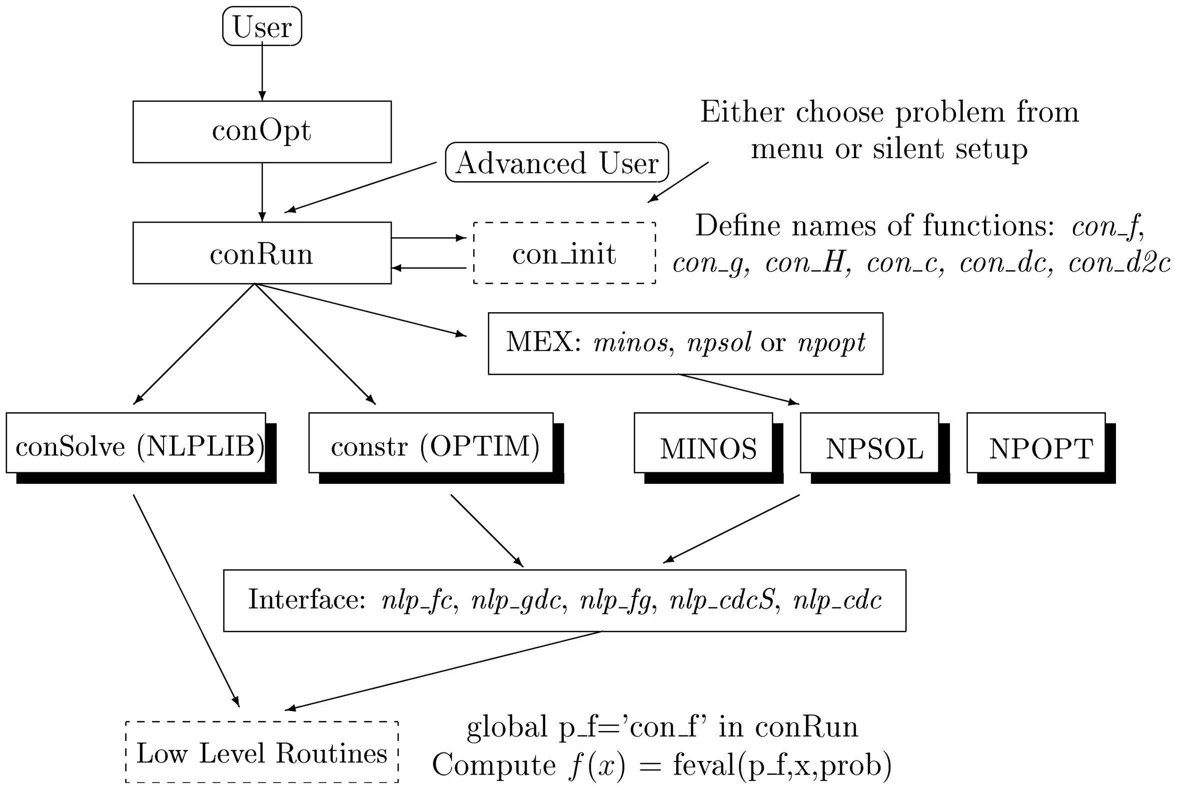
User
conOpt
Advanced User
Either choose problem from menu or silent setup
conRun
con_init
Define names of functions: *con_f*, *con_g, con_H, con_c, con_dc, con_d2c*
MEX: *minos*, *npsol* or *npopt*
conSolve (NLPLIB)
constr (OPTIM)
MINOS
NPSOL
NPOPT
Interface: *nlp_fc*, *nlp_gdc*, *nlp_fg*, *nlp_cdcS*, *nlp_cdc*
Low Level Routines
global p_f='con_f' in conRun
Compute $f(x)$ = feval(p_f,x,prob)

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

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.

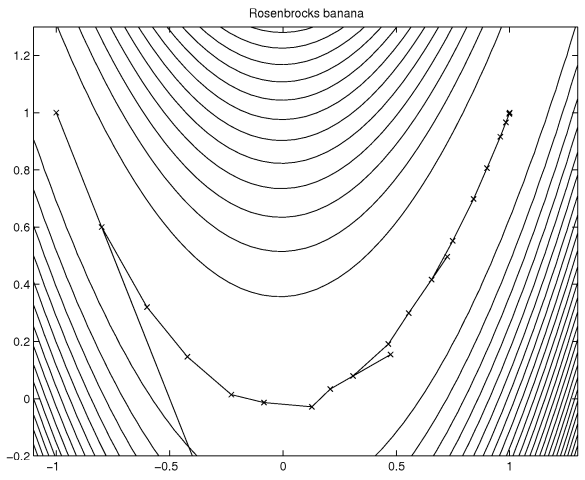

Figure 3: The Rosenbrock banana function with search directions and marks for the line search trials running *ucsolve* using the Newtons method.

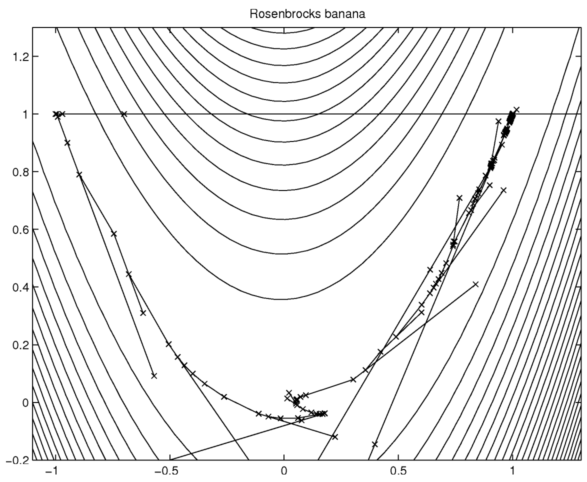

Figure 4: The Rosenbrock banana function with search directions and marks for the line search trials running *ucsolve* using the Fletcher-Reeves conjugate gradient method.

4 The OPERA Toolbox

The MATLAB toolbox **OPERA TB 1.0** was developed as a teaching tool for a course in operations research taught in the autumn of 1994. It is a collection of MATLAB m-files which solves many of the basic optimization problems in operations research and mathematical programming. Currently **OPERA TB 1.0** consists of:

- 11 200 lines of MATLAB code, in directory *opera* and *operdemo*.
- 55 files with algorithms and utilities in the directory *opera*.
- 45 example files in the directory *operdemo*.
- Code compatible with MATLAB 4.2c and MATLAB 5.1.

OPERA TB 1.0 is suitable for teaching basic courses in optimization and operations research. In the following presentation the MATLAB m-file names are given in parenthesis.

The directory structure of **TOMLAB** is displayed in Figure 5.

There are several algorithms implemented for **linear programs** (LP). The standard revised simplex algorithm, as formulated in Goldfarb and Todd [25, page 91], is used to solve the Phase II LP problem (*lpsimp2*). A Phase I simplex strategy which formulates a LP problem with artificial variables is implemented (*lpsimp1*). This routine is using *lpsimp2* to solve the Phase I problem. The dual simplex method [25, pages 105-106], usable when a dual feasible solution is available instead of a primal feasible, is also implemented (*lpdual*).

Two polynomial algorithms for linear programs are implemented. Karmakar's projective algorithm is implemented (*karmark*) using the description in Bazaraa et al. [6, page 386]. There is a choice of update, either according to Bazaraa or the rule by Goldfarb and Todd [25, chap. 9]. The affine scaling variant of Karmakar's method (*akarmark*) is an implementation of the algorithm in Bazaraa [25, pages 411-413]. As the purification algorithm, a modification of the algorithm on page 385 in Bazaraa is used.

Transportation problems are solved using an implementation of the transportation simplex method as described in Luenberger [35, chap 5.4] (*TPsimplx*). Three simple algorithms to find a starting basic feasible solution for the transportation problem are included; the northwest corner method (*TPnw*), the minimum cost method (*TPmc*) and Vogel's approximation method (*TPvogel*). The implementation of these algorithms follows the algorithm descriptions in Winston [45, chap. 7.2].

The implementation of the **Network Programming** algorithms are based on the forward and reverse star representation technique described in Ahuja et al. [4, pages 35-36]. The following algorithms are currently implemented:

- Search for all reachable nodes in a network using a stack approach (*gsearch*). The implementation is a variation of the Algorithm SEARCH in [3, pages 231-233].
- Search for all reachable nodes in a network using a queue approach (*gsearchq*). The implementation is a variation of the Algorithm SEARCH in [3, pages 231-232].
- Find the minimal spanning tree of an undirected graph (*mintree*) with Kruskal's algorithm described in Ahuja et al. [4, page 520-521].
- Solve the shortest path problem using Dijkstra's algorithm (*dijkstra*). A direct implementation of the Algorithm DIJKSTRA in [3, pages 250-251].

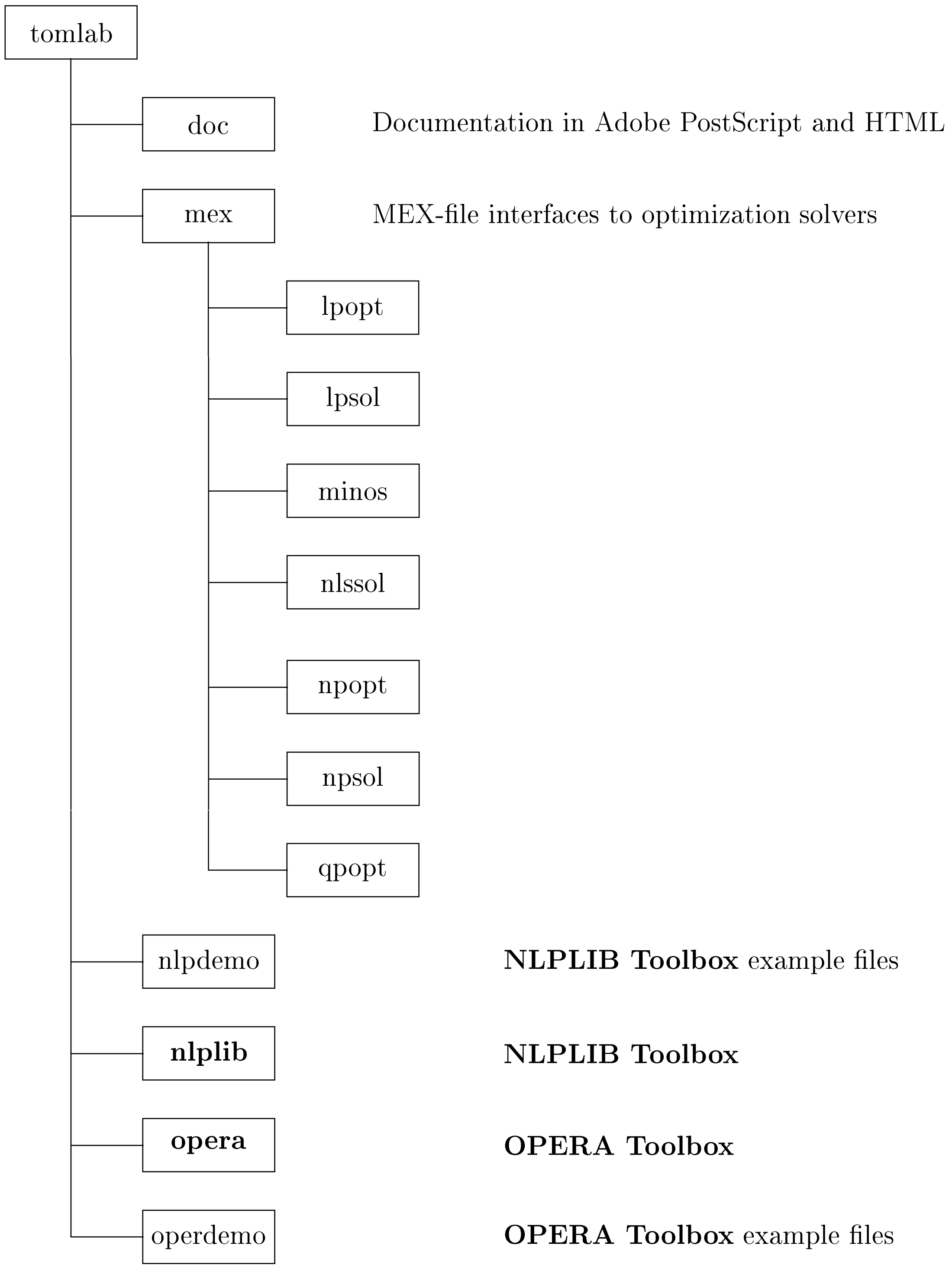
tomlab
doc
Documentation in Adobe PostScript and HTML
mex
MEX-file interfaces to optimization solvers
lpopt
lpsol
minos
nlssol
npopt
npsol
qpopt
nlpdemo
NLPLIB Toolbox example files
nlplib
NLPLIB Toolbox
opera
OPERA Toolbox
operdemo
OPERA Toolbox example files

Figure 5: The directory structure of **TOMLAB**.

- Solve the shortest path problem using a label correcting method (*labelcor*). The implementation is based on Algorithm LABEL CORRECTING in [3, page 260].
- Solve the shortest path problem using a modified label correcting method (*modlabel*). The implementation is based on Algorithm MODIFIED LABEL CORRECTING in [3, page 262], including the heuristic rule discussed to improve running time in practice.
- Solve the maximum flow problem using the Ford-Fulkerson augmenting path method (*maxflow*). The implementation is based on the algorithm description in Luenberger [35, pages 144-145].
- Solve the minimum cost network flow problem (MCNFP) using a network simplex algorithm (*NWsimplx*). The implementation is based on Algorithm network simplex in Ahuja et al. [4, page 415].
- Solve the symmetric traveling salesman problem using Lagrangian relaxation and the subgradient method with the Polyak rule II (*salesman*), an algorithm by Held and Karp [26].

To solve mixed linear inequality integer programs two algorithms are implemented. The first implementation (*branch*) is a branch and bound algorithm from Nemhauser and Wolsey [38, chap. 8]. The second implementation (*cutplane*) is a cutting plane algorithm using Gomory cuts. Both routines are using linear programming routines in the toolbox **OPERA TB 1.0** (*lpsimp1*, *lpsimp2*, *lpdual*) to solve relaxed subproblems. Balas method for 0/1 integer programs restricted to integer coefficients is implemented in the routine *balas* [27].

Two simple examples of dynamic programming are included. Both examples are from Winston [45, chap. 20]. Forward recursion is used to solve an inventory problem (*dpinvent*) and a knapsack problem (*dpknap*).

The usage of Lagrangian relaxation techniques is exemplified by the routine *ksrelax*, which solves integer linear programs with linear inequality constraints and upper and lower bounds on the variables. The problem is solved by relaxing all but one constraint and hence solving simple knapsack problems as subproblems in each iteration. The algorithm is based on the presentation in Fischer [18], using subgradient iterations and a simple line search rule. Lagrangian relaxation is also used by the symmetric travelling salesman solver, see [30]. Also a routine to draw a plot of the relaxed function is included.

5 The NLPLIB Toolbox

The development of **NLPLIB TB 1.0** 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 1.0** 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 1.0** 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;

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 6.

The routine *ucSolve* implements a prototype algorithm for unconstrained optimization with simple bounds on the parameters. It includes several of the most popular search step methods for unconstrained optimization. Bound constraints are treated as described in Gill et al.[24]. The search step methods for **unconstrained optimization** 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;
- A structural trust region algorithm combined
  with an initial trust region radius algorithm.

The first eight methods are implemented as part of the prototype algorithm *ucSolve*. For the Newton and the quasi-Newton methods the code is using a subspace minimization technique to handle rank problems, see Lindström [34]. The quasi-Newton codes also use safe guarding techniques to avoid rank problem in the updated matrix.

The structural trust region algorithm *sTrustR* is based on the algorithms in [12] and [42], but using the same framework as *ucSolve* and the same treatment of problems with bound constraints.

For general **nonlinear problems with nonlinear constraints** a sequential quadratic programming (SQP) method by Schittkowski [43] is implemented in the routine *conSolve*.

Quadratic programs (**qp**) are solved with a standard active set method [35], 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 1.0 includes two algorithms for solving **qp** with equality constraints; a null space method (*qpe*) and Lagrange's method *qplm*).

The line search algorithm *linesrch* used by the solvers *conSolve*, *lsSolve* and *ucSolve* is a modified version of the algorithm by Fletcher [20, chap. 2]. The use of quadratic (*intpol2*) and cubic interpolation (*intpol3*) is possible in the line search algorithm. For more details, see [29].

6 Nonlinear Least Squares Algorithms

In **NLPLIB TB 1.0** the prototype nonlinear least squares algorithm *lsSolve* treats problems with bound constraints in a similar way as the routine *ucSolve* described in Section 5. In Figure 6 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 [34]. 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 [5].
- The Fletcher-Xu hybrid method [19].
- The Huschens TSSM method [31].

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

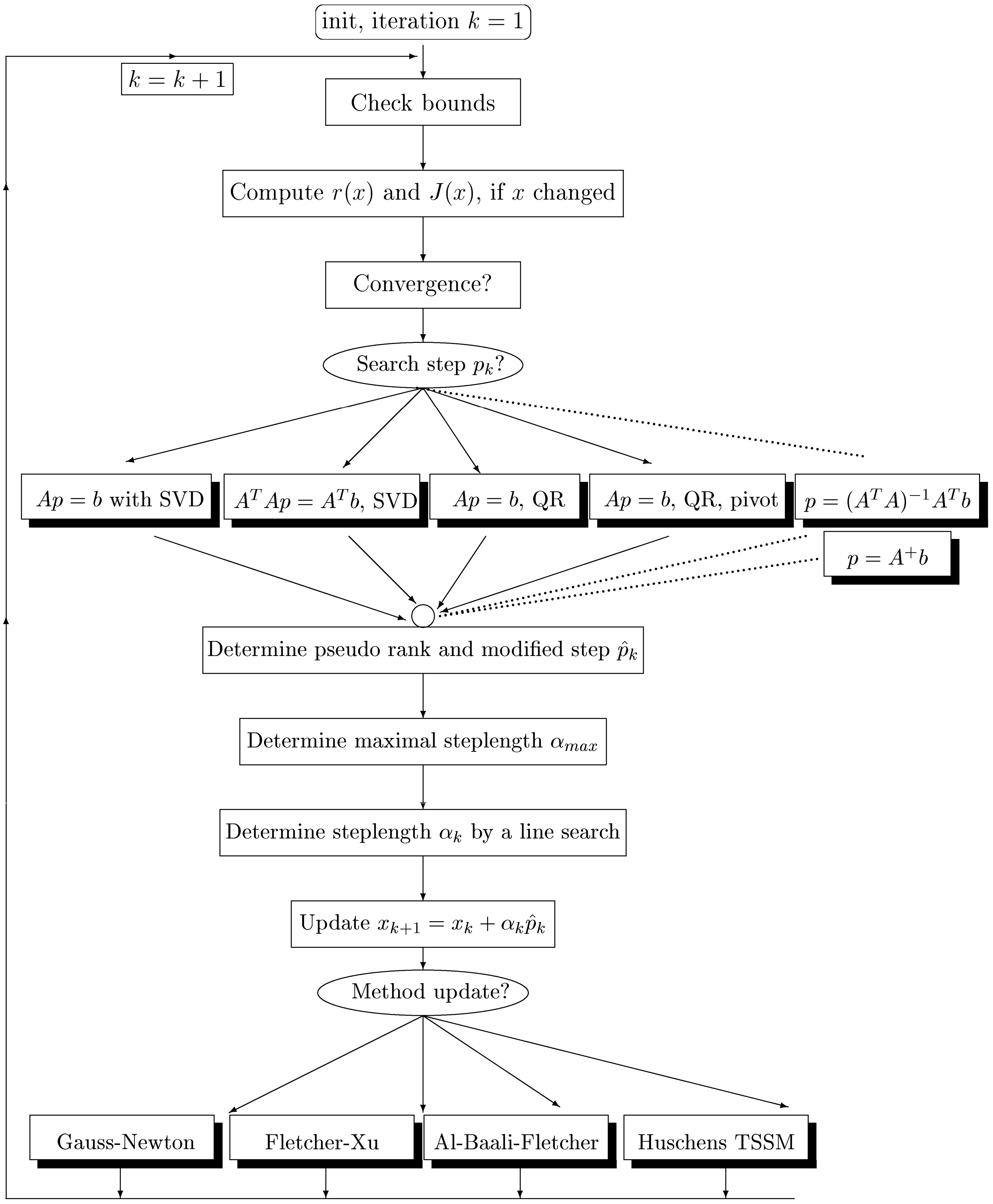

Figure 6: The **NLPLIB TB 1.0** prototype routine for nonlinear least squares

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.

7 Installation of TOMLAB

TOMLAB is distributed either as a file tomlab.tar.gz for UNIX systems or a file tomlab.zip for PC systems. A date could be added to the file name. The file system on the packed file is the tree showed in Figure 5. Some directories may not be included.

The important files for the installation is stored in the top level directory *tomlab*.

The file **tomlab.m** has instructions about the unpacking of the distribution and pointers to the relevant files to continued reading.

The file **contents.m** has a description of all directories in **TOMLAB**.

The file **startup.m** is the startup file for MATLAB, which should be edited to set the correct absolute paths on the machine. Instructions for the editing is included.

The file *tomlab.bib* is a file with all references for TOMLAB and its toolboxes. The format is the *bibtex* data base format.

8 Conclusions

TOMLAB may be used for computer based learning in optimization courses and in computer exercises. **TOMLAB** is currently used in Sweden at Mälardalen University and Uppsala University. Now introduced at Umeå University and at Linköping University.

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

TOMLAB is a flexible tool, with both menu programs and driver routines. It is also a powerful environment for solving optimization problems, with an increasing list of callable solvers.

8.1 Future work

To test new algorithms a good test bench is needed. Standard is the CUTE set of test problems [9]. CUTE gives an easy access to a huge set of test problems. Moreover, CUTE has a set of MATLAB m-files and MEX-file routines which can be used to interface with **TOMLAB**.

It is also desirable to develop an interface to one or more of the modeling languages. The AMPL system [21] has already MEX-file interface routines that communicate with MATLAB on UNIX systems [23]. This should make it easy to interface to **TOMLAB**.

We will develop interfaces to CUTE and AMPL as well as a more complete set of MEX-file interfaces.

The graphics and menus should be improved. A graphical user interface (GUI) for nonlinear programming is being developed. Also plotting routines to follow the decrease of the objective function values and estimate the convergence rate during the iterations. For least squares problems a routine that plots the estimated model against data is developed. We would also like to include more routines for network programs and integer programs. Later we will work on a mixed integer nonlinear optimization (MINLP) solver.

References

[1] *SINDdata. Presentation and Handledning.* Stockholm, Sweden, 1985.

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

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

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

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

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

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

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

[9] 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.

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

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

[12] 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.

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

[14] Erik Dotzauer and Kenneth Holmström. Models for short-term production planning of cogeneration plants. Research and Reports Opuscula ISSN 1400-5468, ISRN HEV-BIB-OP–21-SE, Mälardalen University, Västerås, Sweden, 1997.

[15] Erik Dotzauer and Kenneth Holmström. Optimal Scheduling of Cogeneration Plants. Technical Report IMa-TOM-1997-4, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997. Presented at the 16th International Symposium on Mathematical Programming, Lausanne, Switzerland, August 24-29, 1997.

[16] Arne Stolbjerg Drud. Interactions between nonlinear programing and modeling systems. *Mathematical Programming, Series B*, 79:99–123, 1997.

[17] 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.

[18] Marshall L. Fisher. An Application Oriented Guide to Lagrangian Relaxation. *Interfaces 15:2*, pages 10–21, March-April 1985.

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

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

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

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

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

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

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

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

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

[28] Kenneth Holmström. *Algorithms for Equilibrium Analysis in Solution Chemistry.* PhD thesis, Institute of Information Processing, Umeå University, Sweden, 1988.

[29] Kenneth Holmström. NLPLIB TB 1.0 - A MATLAB Toolbox for Nonlinear Optimization and Parameter Estimation. Technical Report IMa-TOM-1997-2, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997.

[30] 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.

[31] 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.

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

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

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

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

[36] C. B. Moler. MATLAB — an interactive matrix laboratory. Technical Report 369, Department of Mathematics and Statistics, University of New Mexico, 1980.

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

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

[39] Jöran Petersson and Kenneth Holmström. Fitting of Exponential Sums to Empirical Data. Technical Report IMa-TOM-1997-5, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997. Presented at the 16th International Symposium on Mathematical Programming, Lausanne, Switzerland, August 24-29, 1997.

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

[41] Raymond P. Polivka and Sandra Pakin. *APL: The Language and Its Usage.* Prentice Hall, Englewood Cliffs, N. J., 1975.

[42] 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.

[43] 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.

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

[45] Wayne L. Winston. *Operations Research: Applications and Algorithms.* International Thomson Publishing, Duxbury Press, Belmont, California, 3rd edition, 1994.