Constrained Nonlinear Optimization with

EASY-OPT*Express*

- User's Guide Version 1.0, 2009 -

Prof. K. Schittkowski
Department of Computer Science
University of Bayreuth

EASY-OPT*Express* is an interactive software system to solve nonlinear constrained optimization problems, i.e., to minimize an objective function subject to equality or inequality constraints. The underlying numerical algorithm is an implementation of a sequential quadratic programming method, i.e., the code NLPQLP of the author [13, 19]. Model functions are defined in a modeling language called PCOMP and are interpreted and evaluated during runtime. It is assumed that all nonlinear functions are differentiable. Gradients are evaluated automatically. **EASY-OPT***Express* is particularly useful for classroom exercises of optimization courses or to become familiar with optimization routines before starting a *real life* implementation. The mathematical theory of the considered algorithm is described briefly and the usage of the codes is outlined. The user interface is implemented in form of a database under Microsoft Access running under Windows XP or higher. A royalty free runtime version is included.

Important Notes:

1. Trademarks:

   Windows, Microsoft are registered trademarks of Microsoft Corp.
   INTEL is a trademark of Intel Corporation

2. Copyrights:

3. Disclaimer:

Contents

1 Introduction

We proceed from the following mathematical model which describes a general nonlinear programming problem with constraints,

$$\begin{array}{rl} & \min f(x) \\ & g_j(x) = 0 \quad , \; j = 1, ..., m_e \\ x \in I\!R^n : & g_j(x) \geq 0 \quad , \; j = m_e + 1, ..., m \\ & x_l \leq x \leq x_u \end{array} \tag{1}$$

We can imagine, for example, that the objective function is the weight of a mechanical structure that is to be minimized subject to sizing and shape variables, and that the constraints impose limitations on structural response quantities, e.g. upper bounds for stresses or displacements under static loads.

Now we assume that $f(x)$ and $g_1(x)$, ..., $g_m(x)$ are continuously differentiable on the whole $I\!R^n$. To simplify the subsequent notation, we omit the lower and upper bounds on the variables in the subsequent sections.

Nonlinear programming algorithms are usually available in form of subroutines. To adapt a given problem to the code used, model functions and their gradients must be implemented 'by hand', and a main program must be written for calling the optimization routine and for defining all parameters and arrays needed to start the iterative process. This approach is time-consuming and susceptible to programming errors at least for unexperienced users.

In this documentation, we describe a program that reads formatted input data and nonlinear functions in a Fortran-like language, and executes a widely used nonlinear optimization algorithm, the SQP code NLPQLP of the author [19]. Thus, the program is particularly useful for classroom exercises of optimization courses or to become familiar with optimization routines before starting a *real life* implementation.

In order to understand optimization algorithms, i.e., their underlying structure and the results achieved, a brief introduction into mathematical programming theory is presented in Section 2. It is sufficient to consider only optimality criteria which contain all information to analyze a numerical solution. The investigation of the optimality criteria helps to understand the results, e.g., to learn whether an achieved answer can be accepted or not or to get an idea on the final accuracy and the sensitivity of the solution. Moreover, the role of the multipliers is illustrated which are also computed and displayed together with the optimization variables at termination.

The basic strategy of any optimization method is to approximate the given nonlinear problem by another problem that can be solved then much easier. The most efficient strategy leads to successive generation of quadratic programming problems which must be solved then by standard techniques. The resulting algorithm is called

sequential quadratic programming or SQP method and is described in Section 3 in detail. SQP methods are the most frequently used tools today to solve smooth optimization problems, if there are no additional mathematical structures permitting the usage of special purpose techniques.

The main program which reads all input data, evaluates function and gradient values, executes the SQP code NLPQLP, and prepares output data, is called NLPQLADM. To run this program, two input files are required. One contains all numerical data, e.g., number of variables and constraints, starting point, optimization parameters, etc., and another one must contain the problem functions in a special syntax which is similar to Fortran. This file is parsed and an intermediate code is produced needed to start the optimization program. Gradients of nonlinear problem functions are evaluated automatically during run time. A brief outline of the modeling language PCOMP is presented in Section 4, and a summary of all possible error messages is listed in an appendix. The organization of NLPQLADM is described in Section 5.

The *easy-to-use* interface facilitates the execution of the optimization program NLPQLADM.EXE. **EASY-OPT**^{Express} is a database implemented under Microsoft Access 2007, and comes with the royalty-free runtime version. A brief outline of the system is given in Section 6. The usage is described in more detail by an additional context sensitive help file EASY_OPT.HLP.

2 Optimality Criteria

It is outside the scope of this paper to present a detailed introduction into the mathematical theory behind the nonlinear programming problem (1). To get at least an impression on the basic ideas behind optimization algorithms and to learn the most important terms, a brief description of the mathematical optimality criteria is given. We need these criteria to understand the optimization algorithms and to be able to analyze their results.

First we have to specify the notation used in this paper. We denote by

$$\nabla f(x) := (\frac{\partial}{\partial x_1} f(x), \ldots, \frac{\partial}{\partial x_n} f(x))^T$$

the gradient of a differentiable function $f(x)$ for $x \in I\!R^n$. In case of doubts, we add the index x to ∇ to indicate that the differentiation is to be performed only with respect to the x-variables. Moreover, we write the Hessian matrix of a twice differentiable function $f(x)$ with respect to $x \in I\!R^n$ in the form

$$\nabla^2 f(x) := \left(\frac{\partial^2}{\partial x_i \partial x_j} f(x) \right)$$

Again an index x may insure that we consider only the differentiation with respect to x.

Next we define by

$$I(x) := \{j : g_j(x) = 0, m_e < j \leq m\}$$

the set of active constraints for any feasible x, i.e., any x that satisfies all restrictions.

The most important tool to understand the optimality criteria, is the so-called Lagrange function

$$L(x, u) := f(x) - \sum_{j=1}^{m} u_j g_j(x)$$

which is defined for $x \in I\!R^n$ and $u = (u_1, \ldots, u_m)^T$, and which describes a linear combination of the objective function and the constraints. The coefficients $u_j, j = 1, \ldots, m$, are called the Lagrange multipliers of (1).

Now we are able to formulate optimality criteria. Since the required assumptions differ, we distinguish as usual between necessary and sufficient conditions. In the first case, we need an assumption called *constrained qualification*, which means that for a feasible x, the gradients of active constraints, i.e., the set $\{\nabla g_j(x) : j \in I(x)\}$, are linearly independent.

Theorem: Let f and g_j for $j = 1, .., m$ be twice continuously differentiable functions, $x^\star$ be a local minimizer of (1) and the constrained qualification be satisfied in

$x^\star$. Then there is a $u^\star \in I\!R^m$ so that the following conditions are satisfied:

$$\begin{array}{ll} \text{a)} & u_j^\star \geq 0 \text{ for } j = m_e + 1, \ldots, m, \\ & \nabla_x L(x^\star, u^\star) = 0, \\ & u_j^\star g_j(x^\star) = 0 \text{ for } j = m_e + 1, \ldots, m, \\ \text{b)} & s^T \nabla_x^2 L(x^\star, u^\star) s \geq 0 \text{ for all } s \in I\!R^n \text{ with } \nabla g_j(x^\star)^T s = 0, \\ & j \in \{1, \ldots, m_e\} \cup I(x^\star) \end{array}$$

It is possible to prove the opposite direction without the regularity condition *constraint qualification* subject to some stronger statement, i.e., a strict local minimizer.

Theorem: Let f and g_j for $j = 1, .., m$ be twice continuously differentiable functions, $x^\star \in I\!R^n$ be feasible with respect to (1) and $u^\star \in I\!R^n$ a multiplier vector with

$$\begin{array}{ll} \text{a)} & u_j^\star \geq 0 \text{ for } j = m_e + 1, \ldots, m, \\ & \nabla_x L(x^\star, u^\star) = 0, \\ & u_j^\star g_j(x^\star) = 0 \text{ for } j = m_e + 1, \ldots, m, \\ \text{b)} & s^T \nabla_x^2 L(x^\star, u^\star) s > 0 \text{ for all } s \in I\!R^n, s \neq 0, \text{ with } \nabla g_j(x^\star)^T s = 0, \\ & j \in \{1, \ldots, m_e\} \text{ and } j = m_e + 1, \ldots, m \text{ with } u_j^\star > 0. \end{array}$$

Then $x^\star$ is an isolated local minimizer of f, i.e., there is a neighborhood $U(x^\star)$ of $x^\star$, so that $f(x^\star) < f(x)$ for all $x \in U(x^\star)$, $x \neq x^\star$.

The condition, that the gradient of the Lagrange function vanishes at an optimal solution, is called the *Karush-Kuhn-Tucker condition* of KKT condition of (1). In other words, the gradient of f is a linear combination of gradients of active constraints,

$$\nabla f(x^\star) = \sum_{j \in I(x^\star)} u_j^\star \nabla g_j(x^\star)$$

The complementary slackness condition $u_j^\star g_j(x^\star) = 0$ together with the feasibility of $x^\star$ guarantees, that only the active constraints, i.e., the interesting ones, contribute a gradient in the above sum. Either a constraint is satisfied by equality or the corresponding multiplier value is zero.

The Karush-Kuhn-Tucker condition can be computed within an optimization algorithm, if suitable multiplier estimates are available, and serves as a stopping condition. However, the second order condition b) can only be evaluated numerically, if second derivatives are available. The condition is required in the optimality criteria, to be able to distinguish between a stationary point and a local minimizer.

Proofs of the theorems are found in any textbook on nonlinear programming, e.g. in Gill, Murray and Wright [4] or Spellucci [20]. An elementary outline of the optimality conditions and a geometric interpretation is found in Papalambros and Wilde [8].

3 Sequential Quadratic Programming Algorithms

The sequential quadratic programming or SQP method is the standard general purpose tool for solving smooth nonlinear optimization problems under the following assumptions:

- The problem is not too big.
- The functions and gradients can be evaluated with sufficiently high precision.
- The problem is smooth and well-scaled.

The mathematical convergence and the numerical performance properties of SQP methods are very well understood now and are published in so many papers, that only a few can be mentioned here. Theoretical convergence is investigated in Han [5, 6], Powell [9, 10], Schittkowski [12], e.g., and the numerical comparative studies of Schittkowski [11] and Hock, Schittkowski [7] show their superiority over other mathematical programming algorithms under the above assumptions.

The key idea is to approximate also second order information to get a fast final convergence speed. Thus we define a quadratic approximation of the Lagrange function $L(x,u)$ and an approximation of the Hessian matrix $\nabla_x^2 L(x_k,u_k)$ by a so-called quasi-Newton matrix B_k. Then we get the subproblem

$$\begin{array}{rll} & \min \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d & \\ d \in I\!R^n: & \nabla g_j(x_k)^T d + g_j(x_k) = 0 & , \; j = 1,\ldots,m_e \\ & \nabla g_j(x_k)^T d + g_j(x_k) \geq 0 & , \; j = m_e+1,\ldots,m \end{array} \tag{2}$$

To stabilize the algorithm particularly when starting from a poor initial guess x_0, and to ensure convergence, an additional line search is performed, i.e. a steplength computation to accept a new iterate $x_{k+1} := x_k + \alpha_k d_k$ for an $\alpha_k \in (0,1]$ only if x_{k+1} satisfies a descent property with respect to a solution d_k of (2). Following the approach of Schittkowski [12], e.g., we need also a simultaneous line search with respect to the multiplier approximations called v_k and define $v_{k+1} := v_k + \alpha_k(u_k - v_k)$ where u_k denotes the optimal Lagrange multiplier of the quadratic programming subproblem (2).

The line search is performed with respect to a merit function

$$\psi_k(\alpha) := \phi_{r_k}(x_k + \alpha d_k, v_k + \alpha(u_k - v_k))$$

and

$$\begin{array}{rll} \phi_r(x,v) := & f(x) - \sum_{j=1}^{m_e}(v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2) & \\ & - \sum_{j=m_e+1}^{m} \left\{ \begin{array}{ll} (v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2), & \text{if } g_j(x) \leq v_j/r_j \\ \frac{1}{2} v_j^2/r_j, & \text{otherwise} \end{array} \right. & \end{array}$$

where $r = (r_1, \ldots, r_m)^T$. We should note here that also other concepts, i.e. other merit functions are found in the literature. We initiate a subiteration starting with $\alpha = 1$ and perform a successive reduction combined with a quadratic interpolation of $\psi_k(\alpha)$, until for the first time, a the stopping condition of the form

$$\psi_k(\alpha) \leq \psi_k(0) + \mu\alpha\psi_k'(0)$$

is satisfied, where we must be sure that $\psi_k'(0) < 0$, of course. To guarantee this condition, the penalty parameter r_k must be evaluated by a special formula, see Schittkowski [12].

The update of the matrix B_k can be performed by standard techniques known from unconstrained optimization. In most cases, the BFGS-method is applied, a numerically simple rank-2 correction starting from the identity or any other positive definite matrix. Only the difference vectors $x_{k+1} - x_k$ and $\nabla_x L(x_{k+1}, u_k) - \nabla_x L(x_k, u_k)$ are required. Under some safeguards it is possible to guarantee that all matrices B_k are positive definite.

Among the most attractive features of SQP methods is the superlinear convergence speed in the neighborhood of a solution given by

$$\| x_{k+1} - x^* \| \leq \gamma_k \| x_k - x^* \|$$

where γ_k is a sequence of positive numbers converging to zero and x^* an optimal solution.

To understand this convergence behavior, replace B_k by the true Hessian of the Lagrangian function and consider only equality constraints. Then it is very easy to see that an SQP method is nothing else than Newton's method for solving the nonlinear system of $n + m$ equations in $n + m$ unknowns given by the KKT conditions. This result can be extended to inequality constraints as well. Then we get immediately the quadratic convergence behavior and, if we replace B_k again by its approximation, the weaker superlinear convergence rate as proved in the references mentioned above.

4 The Modeling Language PCOMP

The symbolic input of nonlinear functions was developed for a package of automatic differentiation routines called PCOMP, see Dobmann, Liepelt and Schittkowski [2]. Basically, the language is a subset of Fortran with a few extensions. In particular the declaration and executable statements must satisfy the usual Fortran input format, i.e. must start at the 7th position or later. Comments beginning with 'C' at the first column, may be included in a program text whenever needed. Statements may be continued on subsequent lines by a continuation mark in the 6th column. Either capital or small letters are allowed.

In contrast to Fortran, however, most variables are declared implicitly by their assignment statements. Only those variables and functions must be declared separately, which are used for automatic differentiation. PCOMP possesses six special constructs to identify program blocks.

* SET OF INDICES
  Definition of index sets that can be used to declare data, variables and functions or to define SUM - or $PROD$ - statements.

* REAL CONSTANT
  Definition of real data, either without index or with one- or two-dimensional index.

* INTEGER CONSTANT
  Definition of integer data, either without index or with one- or two-dimensional index.

* TABLE
  Definition of real data in form of spreadsheet, i.e., for given indices of one or two-dimensional array specified, the corresponding data are to be listed row by row.

* VARIABLE
  Declaration of variables either with or without index, with respect to which automatic differentiation is to be performed.

* FUNCTION $\langle$ identifier $\rangle$
  Declaration of functions either with or without index, for which function and gradient values are to be evaluated. The subsequent statements must address a numerical value to the function identifiers.

* END
  End of the program.

The order of the above program blocks is obligatory, but they may be repeated whenever desirable. The END-statement must be the final command. The statements within the program blocks are very similar to usual Fortran notation and must satisfy the following guidelines:

Constant data: For defining real numbers either in analytical expressions or within the special constant data definition block, the usual Fortran convention can be used. In particular the F-, E- or D-format is allowed.

Identifier names: Names of identifiers, e.g., variables and functions, index sets and constant data have to follow the Fortran syntax rules. Up to six alphanumerical characters are allowed.

Index sets: Index sets are required for the SUM- and $PROD$-expressions and for defining indexed data, variables and functions. They can be defined in two alternative ways:

1. Range of indices, e.g.
$$IND1 = 1..27$$
2. Set of indices, e.g.
$$IND2 = 3, 1, 17, 27, 20$$
3. Computed index sets, e.g.
$$IND3 = 5 * I + 100, I \ \ in \ \ IND1$$

Assignment statements: As in Fortran, assignment statements are used to address a numerical value to an identifier, which may be either the name of the nonlinear function that is to be defined, of of an auxiliary variable that is used in subsequent expressions, e.g.,

$$\begin{aligned} R1 &= X1 * X4 + X2 * X4 + X3 * X2 - 11 \\ R2 &= X1 + 10 * X2 - X3 + X4 + X2 * X4 * (X3 - X1) \\ F &= R1 * *2 + R2 * *2 \end{aligned}$$

Analytical expressions: An analytical expression is, as in Fortran, any allowed combination of constant data, identifiers, elementary or intrinsic arithmetic operations and the special SUM- and $PROD$-statements. Elementary operations are

$$+ \ , \ - \ , \ * \ , \ / \ , \ **$$

and the allowed intrinsic functions are

$$SIN\ ,\ COS\ ,\ TAN\ ,\ ASIN\ ,\ ACOS\ ,\ ATAN\ ,$$
$$SINH\ ,\ COSH\ ,\ TANH\ ,\ ASINH\ ,\ ACOSH\ ,\ ATANH\ ,$$
$$ABS\ ,\ EXP\ ,\ LOG\ ,\ LOG10\ ,\ SQRT$$

Alternatively, the corresponding double precision Fortran names for intrinsic functions possessing an initial 'D' can be used as well. Brackets are allowed to combine groups of operations. Possible expressions are e.g.

$$5 * DEXP(-Z(I))$$

or

$$LOG(1 + SQRT(C1 * F1) * *2)$$

SUM **- and** *PROD***-expressions:** Sums and products over predetermined index sets are formulated by SUM- and $PROD$-expressions, where the corresponding index and the index set must be specified, e.g. in the form

$$F = 100 * PROD(X(I) * *A(I), I\ \ in\ \ INDA)$$

In the above example, $X(I)$ might be a variable vector defined by an index set, and $A(I)$ an array of constant data.

Control statements: To control the execution of a program, the conditional statements

$$IF\ \langle condition \rangle\ THEN$$
$$\langle statements \rangle$$
$$ENDIF$$

or

$$IF\ \langle condition \rangle\ THEN$$
$$\langle statements \rangle$$
$$ELSE$$
$$\langle statements \rangle$$
$$ENDIF$$

can be inserted into a program. Conditions are defined as in Fortran by the comparative operators EQ, NE, LE, LT, GE, GT, which can be combined using brackets and the logical operators AND, OR and NOT, e.g.

$$Y1 = (X3 - X2) * (X5 - X2) * (X6 - X2)$$
$$IF\ ((Y1.LT.EPS)\ .AND.\ (Y1.GT. - EPS))\ THEN$$
$$Y1 = EPS$$
$$ENDIF$$

Whenever indices are used within arithmetic expressions, it is possible to insert polynomial expressions of indices from a given set. However functions must be treated in a particular way. Because of the internal structure of the reverse algorithm and the design goal to generate short, efficient Fortran codes, indexed function names can be used only in exactly the same way as defined. In other words, if a set of functions is declared, e.g., by

$$* \, FUNCTION \; F(I), \;\; I \; IN \; INDEX$$

then only accesses to $F(I)$ are allowed, not to $F(1)$ or $F(J)$, for example. For a very similar reason it is not allowed to have any variable names on the left and right hand side of an arithmetic expression at the same time and PCOMP will report an error message in this case. Thus a statement like

$$S \; = \; S \; + \; 2 * X1$$

is forbidden.

On the other hand it is allowed to pass variable values from one function block to the other. However the user must be aware of a possible failure if in the calling program, the evaluation of a gradient value is not required for a function, for which a certain variable is defined that is used also in subsequent blocks. It should be noted that DO-expressions are not allowed, also subroutine calls are forbidden.

However one should be very careful when using the conditional statement IF. Possible traps that prevent a correct differentiation are reported in Fischer [3], and are to be illustrated by an example. Consider the function $f(x) = x^2$ for $n = 1$. A syntactically correct formulation would be:

$$\begin{aligned}
&IF \; X = 1 \; THEN \\
&F \; = \; 1 \\
&ELSE \\
&F \; = \; X * *2 \\
&ENDIF
\end{aligned}$$

In this case PCOMP would try to differentiate both branches of the conditional statement. If X is equal to 1, the derivative value of F is zero, otherwise equal to $2 * X$. Obviously we get a wrong answer for $X = 1$. This is a basic drawback for all automatic differentiation algorithms of the type we are considering.

Examples in form of complete PCOMP-programs are listed in reference Dobmann, Liepelt and Schittkowski [2] and in the subsequent sections in form of nonlinear programming test cases.

5 Program Organization

The nonlinear programming problem is solved by the code NLPQLP, see Schittkowski [13, 19], which has been extended over a period of 30 years, e.g. to run under a distributed system or to allow uphill search directions. If desired by the user, the output of NLPQL is directed to a file named ⟨name⟩.OUT.

To run NLPQLADM.EXE as a stand-alone code, one has to provide two input files with the names NLPQLADM.DAT and ⟨name⟩.FUN, where ⟨name⟩ denotes any name to identify the optimization problem to be solved. The file with extension .FUN contains the formulation of the problem functions in a Fortran-like language which was developed for the automatic differentiation code PCOMP, see Dobmann, Liepelt and Schittkowski [2] or the previous section. These two files are created automatically by the user interface of **EASY-OPT**^{Express} and the code NLPQLADM.EXE is started from the interface.

The first file with extension DAT contains problem parameters, data, solution tolerances, initial variable values etc., which are necessary to start the optimization algorithm. The input format is described subsequently in more detail. It is important here to note that the user must follow the input guidelines very carefully. It is e.g. not allowed to omit or interchange lines or column positions. For an experienced user it should be possible to integrate the software into his own environment related to the domain of application, and to generate this input file by his own software.

The interactive user interface coming with **EASY-OPT**^{Express} facilitates the usage of NLPQLADM. Input of data is required in form of windows and masks, and the system generates the input files for the numerical algorithms automatically. After termination of on optimization run, results are read and stored in the database.

The two input files with extensions FUN and DAT must fit together. The following rules apply:

- Variable names are declared within the variable block of the model declaration file with extension FUN. The numbers of variables on this file must coincide with the number of variables, n, defined in the file NLPQLADM.DAT.
- The number of nonlinear constraints m must also coincide on both files. First the objective function, then the m_e equality constraints, if available, and subsequently the $m - m_e$ inequality constraints must be defined in ⟨name⟩.FUN, either individually or by index sets.
- Any other functions are not allowed to be defined.
- Index sets and constant data can be defined in addition, if required for defining model functions.

Several output files are generated by the optimization programs. They serve to adapt the codes to a special environment, e.g. the interactive user interface EASY-OPT. Some of them are optional.

1. ⟨name⟩.RES: Numerical results, e.g. optimal variable and function values. The format depends on the system executed, and is described later.

2. ⟨name⟩.OHF: Function values for displaying optimization history, i.e., iteration numbers and objective function values in two columns.

3. ⟨name⟩.OHR: Constraint violation values for displaying optimization history, i.e., iteration numbers and constraint violations in two columns.

4. ⟨name⟩.OUT: Optional output channel of the SQP code NLPQLP, not supported by the GUI of **EASY-OPT**^{Express} .

Data input on file NLPQLADM.DAT:

Line	Format	Name	Description
1	A80		Path to the model function file ⟨name⟩.FUN, but without extension FUN, for retrieving equations and for creating output on files with different extensions.
2	10X,A60	INFO	Arbitrary information string.
4	10X,I5	ISCRN	Display of optimization output on screen (ISCRN>0) or on a file with name ⟨name⟩.OUT (ISCRN=0).
3	10X,G20.10	ACCQP	Accuracy for solving the quadratic programming subproblem, e.g., 1.E-14.
5	10X,G20.10	ACC	Desired final accuracy, e.g. 1.E-7.
6	10X,I5	MAXIT	Maximum number of iterations.
7	10X,I5	MAXFUN	Maximum number of function calls in the line search subalgorithm, e.g. 20.
8	10X,I5	IPRINT	Specification of the desired output level.
		IPRINT=0:	No output of the program.
		IPRINT=1:	Only a final convergence analysis is given.
		IPRINT=2:	One line of intermediate results is printed in each iteration.
		IPRINT=3:	More detailed information is printed in each iteration step, e.g. variable, constraint and multiplier values.

Line	Format	Name	Description
9	10X,I5	N	Number of variables. N must be greater than 1 and not greater than 200.
10	10X,I5	M	Number of equality and inequality constraints without bounds, not greater than 1,000.
11	10X,I5	ME	Number of equality constraints, not greater than M.
	.....		Repeat lines 12 to 14 for i=1, ..., N:
12	10X,G20.10	X(I)	Input of i-th coefficient of initial variable value x_0.
13	10X,G20.10	XL(I)	Input of i-th coefficient of lower bound x_l. XL(I) must not be greater than X(I).
14	10X,G29.15	XU(I)	Input of i-th coefficient of upper bound x_u. XU(I) must not be smaller than X(I).

Output of results on file ⟨name⟩.RES:

Format	Name	Description
1X,4I10	IFAIL	Termination code according to the following list:
	IFAIL=0:	The optimality conditions are satisfied.
	IFAIL=1:	The algorithm has been stopped after MAXIT iterations.
	IFAIL=2:	The algorithm computed an uphill search direction.
	IFAIL=3:	Underflow occurred when determining a new approximation matrix for the Hessian of the Lagrangian function.
	IFAIL=4:	More than MAXFUN function evaluations are required during the line search algorithm.
	IFAIL=5:	Length of a working array is too short. More detailed error information is obtained with IPRINT > 0.
	IFAIL=6:	There are false dimensions, i.e. M $>$ MMAX, N $\geq$ NMAX, or MNN2 $\neq$ M+N+N+2.
	IFAIL=7:	The search direction is close to zero, but the current iterate is still infeasible.
	IFAIL>10:	The solution of the quadratic or least squares subproblem has been terminated with an error message IFQL > 0 and IFAIL is set to IFAIL=IFQL+10.
	NFUNC	Number of function evaluations.
	NGRAD	Number of gradient evaluations.
	NQUSUB	Number of solutions of the quadratic programming subproblem.
For I=1, ..., N:		
1X,D19.8	X(I)	I-th final variable value.
1X,D19.8	U(I)	Corresponding multiplier value with respect to an active upper or lower bound. If no bounds are active, the value is zero.
1X,D19.8	F	Objective function value.
For J=1,...,M:		
1X,D19.8	G(J)	J-th final constraint value.
1X,D19.8	U(J)	Corresponding multiplier value.
1X,D19.8	CONVIO	Sum of constraint violation.

Format	Name	Description
1X,D19.8	SUMMUL	Sum of multiplier values.

Examples:

We want to illustrate the usage of the program NLPQLADM by several examples. The first problem is test example TP32 of Hock and Schittkowski [7]. We show the input files and the output file TP32.RES.

$$
\begin{aligned}
& & \min & (x_1 + 3x_2 + x_3)^2 + 4(x_1 - x_2)^2 \\
x_1, x_2, x_3 : & & 1 - x_1 - x_2 - x_3 &= 0 \\
& & 6x_2 + 4x_3 - x_1^3 - 3 &\geq 0 \\
& & 0 \leq x_1,\ 0 \leq x_2,\ & 0 \leq x_3
\end{aligned}
$$

First we consider the test example as a nonlinear programming problem without any special structure, and get the following two input files:

TP32.FUN:

```
*     VARIABLE
      X1, X2, X3
C
*     FUNCTION F
      F = (X1 + 3*X2 + X3)**2 + 4*(X1 - X2)**2
C
*     FUNCTION G1
      G1 = 1 - X1 - X2 - X3
C
*     FUNCTION G2
      G2 = 6*X2 + 4*X3 - X1**3 - 3
C
*     END
```

NLPQLADM.DAT:

```
      C:\EASY_OPT\PROBLEMS\TP32
      INFO   = Test problem TP32 of the Hock-Schittkowski collection
      ISCRN  = 1
      ACCQP  = 1.D-14
      ACC    = 1.D-10
      MAXIT  = 40
      MAXFUN = 8
      IPRINT = 2
```

```
N       = 3
M       = 2
ME      = 1
X1      = 0.1
XL1     = 0.0
XU1     = 1000.0
X2      = 0.7
XL2     = 0.0
XU2     = 1000.0
X3      = 0.2
XL3     = 0.0
XU3     = 1000.0
```

TP32.RES:

```
   0        3         3         3
 .00000000D+00
 .00000000D+00
 .92444637D-32
 .40000000D+01
 .10000000D+01
 .00000000D+00
 .10000000D+01
 .00000000D+00
-.20000000D+01
 .10000000D+01
 .00000000D+00
 .00000000D+00
 .60000000D+01
```

Index sets are often useful to define a series of functions or variables. The subsequent example shows, how they can be used together with the formulation of sums of functions. The problem is the unconstrained least squares data fitting problem to minimize

$$f(x) = \sum_{i=1}^{10} (\exp(-x_1 t_i) - \exp(-x_2 t_i) - x_3(\exp(-t_i) - \exp(-10t_i)))^2$$

where $t_i = 0.1i$ and where the variables are bounded below by 0 and above by 10, see test problem TP242 of Schittkowski [14]. Also we want to use the opportunity to show, how constants can be defined over index sets.

TP242.FUN:

```
*     SET OF INDICES
      IND = 1..10
C
*     REAL CONSTANT
      A    = 10
```

```
      T(I) = 0.1*I, I IN IND
      Y(I) = EXP(-T(I)) - EXP(-A*T(I)), I IN IND
C
*     VARIABLE
      X1, X2, X3
C
*     FUNCTION F
      F = SUM((EXP(-X1*T(I)) - EXP(-X2*T(I)) - X3*Y(I))**2, I IN IND)
C
*     END
```

NLPQLADM.DAT:

```
      C:\EASY_OPT\PROBLEMS\TP242
      INFO    = Test problem TP242 of the Schittkowski collection
      ISCRN   = 1
      ACCQP   = 1.D-14
      ACC     = 1.D-10
      MAXIT   = 40
      MAXFUN  = 20
      IPRINT  = 2
      N       = 3
      M       = 0
      ME      = 0
      X1      = 2.5
      XL1     = 0.0
      XU1     = 100.0
      X2      = 10.0
      XL2     = 0.0
      XU2     = 100.0
      X3      = 10.0
      XL3     = 0.0
      XU3     = 100.0
```

The final example shows how constant data can be passed to the function definition part through the $TABLE$-command. The problem is test example TP111 of Hock and Schittkowski [7]:

$$
\begin{array}{rll}
 & \min \sum_{j=1}^{10} \exp(x_j)(c_j + x_j - \log(\sum_{k=1}^{10} \exp(x_k))) & \\
 & \exp(x_1) + 2\exp(x_2) + 2\exp(x_3) + \exp(x_6) + \exp(x_{10}) - 2 & = 0 \\
x \in I\!R^{10}: & \exp(x_4) + 2\exp(x_5) + \exp(x_6) + \exp(x_7) - 1 & = 0 \\
 & \exp(x_3) + \exp(x_7) + \exp(x_8) + 2\exp(x_9) + \exp(x_{10}) - 1 & = 0 \\
 & -100 \le x_i \le 100\,, \quad i = 1, \ldots, 10 &
\end{array}
$$

The constant data for c_j are give below in the function input file, and are stored internally when parsing the functions.

TP111.FUN:

```
*     SET OF INDICES
      IND   = 1..10
      INDG1 = 1,2,3,6,10
      INDG2 = 4,5,6,7
      INDG3 = 3,7,8,9,10
C
*     TABLE A(I), I IN IND
      1  .6089D+01
      2  .17164D+02
      3  .34054D+02
      4  .59140D+01
      5  .24721D+02
      6  .14986D+02
      7  .24100D+02
      8  .10708D+02
      9  .26662D+02
      10 .22179D+02
C
*     REAL CONSTANT
      B1(I)=0, I IN IND
      B1(1)=1.0
      B1(2)=2.0
      B1(3)=2.0
      B1(6)=1.0
      B1(10)=1.0
C
      B2(I)=0, I IN IND
      B2(4)=1.0
      B2(5)=2.0
      B2(6)=1.0
      B2(7)=1.0
C
      B3(I)=0, I IN IND
      B3(3)=1.0
      B3(7)=1.0
      B3(8)=1.0
      B3(9)=2.0
      B3(10)=1.0
C
*     VARIABLE
      X(I), I IN IND
C
*     FUNCTION F
      F1 = SUM(A(I)*EXP(X(I)), I IN IND)
      F2 = SUM(EXP(X(I)), I IN IND)
      F  = -F1 + SUM(EXP(X(J))*(X(J) - LOG(F2)), J IN IND)
C
*     FUNCTION G1
      G1 = -SUM(B1(I)*EXP(X(I)), I IN INDG1) + 2
C
*     FUNCTION G2
      G2 = -SUM(B2(I)*EXP(X(I)), I IN INDG2) + 1
```

```
C
*     FUNCTION G3
      G3 = -SUM(B3(I)*EXP(X(I)), I IN INDG3) + 1
C
*     END
```

6 The Easy-to-Use Interface

The *easy-to-use* interactive system **EASY-OPT**^{Express} facilitates the execution of the optimization program NLPQLADM. The documentation on the mathematical models, the numerical algorithms, the input format of data, and the syntax of the language for defining the model functions, can be retrieved also interactively by the context-sensitive help facility.

The installation of **EASY-OPT**^{Express} is described by some separate installation notes. The database comes with the distribution-free run-time version of Microsoft Access 2007. Plots are generated by GNUPLOT.

EASY-OPT^{Express} requires the input of functions in the syntax of the PCOMP system described in the previous sections. A large number of examples comes with **EASY-OPT**^{Express} taken from the test problem collections [7] and [14].

When defining a model by the input language PCOMP, we have to follow certain guidelines for the declaration of parameters and functions, since the succession in which these items are defined, is essential for the interface between the input file and the executed code. Model functions are defined in the following order:

1. First, the objective function $f(x)$ must be defined where names for the variables $x_i, i = 1, \ldots, n$ must be declared in the VARIABLE section.

2. The subsequent m_e functions are the equality constraints $g_1(x)$, ..., $g_{m_e}(x)$, if they exist.

3. Finally, $m - m_e$ functions for inequality constraints $g_{m_e+1}(x)$, ..., $g_m(x)$ are defined.

4. Any other functions are not allowed.

The constants n, m_e, and m are defined in the data input file and must coincide with the corresponding number of functions. The order and the number of variables must coincide with the number of bounds and starting values defined separately. In addition to variables and functions, a user may insert further real or integer constants in the function input file according to the guidelines of the PCOMP language.

The maximum number of variables is 200, the maximum number of constraints 1,000. Note that variables and functions may be declared over index sets.

A special option of the output command of **EASY-OPT**^{Express} is a display of a so-called *optimization history*. Objective function values and the sum of constraint violations are displayed over the number iterations performed. One iteration is defined by one evaluation of gradients of objective function and all constraints.

,

References

[1] Armijo L. (1966): *Minimization of functions having Lipschitz continuous first partial derivatives,* Pacific Journal of Mathematics, Vol. 16, 1–3

[2] Dobmann M., Liepelt M., Schittkowski K. (1995): *Algorithm 746: PCOMP: A Fortran code for automatic differentiation,* ACM Transactions on Mathematical Software, Vol. 21, No. 3, 233-266

[3] Fischer H. (1991): *Special problems in automatic differentiation,* in: Automatic Differentiation of Algorithms: Theory, Implementation and Application, A. Griewank, G. Corliss eds., SIAM, Philadelphia

[4] Gill P.E., Murray W., Wright M.H. (1981): *Practical Optimization,* Academic Press, New York, London

[5] Han S.-P. (1976): *Superlinearly convergent variable metric algorithms for general nonlinear programming problems* Mathematical Programming, Vol. 11, 263-282

[6] Han S.-P. (1977): *A globally convergent method for nonlinear programming* Journal of Optimization Theory and Applications, Vol. 22, 297-309

[7] Hock W., Schittkowski K. (1981): *Test Examples for Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 187, Springer, Berlin

[8] Papalambros P.Y., Wilde D.J. (1988): *Principles of Optimal Design,* Cambridge University Press

[9] Powell M.J.D. (1978): *A fast algorithm for nonlinearly constraint optimization calculations,* in: Numerical Analysis, G.A. Watson ed., Lecture Notes in Mathematics, Vol. 630, Springer, Berlin

[10] Powell M.J.D. (1978): *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, in: Nonlinear Programming 3, O.L. Mangasarian, R.R. Meyer, S.M. Robinson eds., Academic Press, New York, London

[11] Schittkowski K. (1980): *Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 183 Springer, Berlin

[12] Schittkowski K. (1983): *On the convergence of a sequential quadratic programming method with an augmented Lagrangian search direction,* Mathematische Operationsforschung und Statistik, Ser. Optimization, Vol. 14, 197-216

[13] Schittkowski K. (1985/86): *NLPQL: A Fortran subroutine solving constrained nonlinear programming problems,* Annals of Operations Research, Vol. 5, 485-500

[14] Schittkowski K. (1987): *More Test Examples for Nonlinear Programming,* Lecture Notes in Economics and Mathematical Systems, Vol. 182, Springer, Berlin

[15] Schittkowski K. (2002): *EASY-FIT: A software system for data fitting in dynamic systems*, Structural and Multidisciplinary Optimization, Vol. 23, No. 2, 153-169

[16] Schittkowski K. (2002): *Numerical Data Fitting in Dynamical Systems - A Practical Introduction with Applications and Software*, Kluwer Academic Publishers, Dordrecht, Boston, London

[17] Schittkowski K. (2003): *QL: A Fortran code for convex quadratic programming - user's guide*, Report, Department of Mathematics, University of Bayreuth, 2003

[18] Schittkowski K. (2004): *PCOMP: A modeling language for nonlinear programs with automatic differentiation*, in: *Modeling Languages in Mathematical Optimization*, J. Kallrath ed., Kluwer, Norwell, MA, 349-367

[19] Schittkowski K. (2007): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user's guide*, Report, Department of Computer Science, University of Bayreuth

[20] Spellucci P. (1993): *Numerische Verfahren der nichtlinearen Optimierung,* Birkhäuser, Boston, Basel, Berlin

APPENDIX A: Error Messages of the PCOMP-Parser

PCOMP reports error messages in the form of integer values of the variable IERR and, whenever possible, also line numbers LNUM. The meaning of the messages is listed in the following table. Note that the corresponding text is displayed if the error routine SYMERR is called with parameters LNUM and IERR.

In the version implemented for the parameter estimation codes, an error is reported when starting the execution of a numerical algorithm, i.e., when the parser analyzes the code. The corresponding error code and a line number are displayed and a user should edit the PCOMP code before trying it again.

Format	Name	Description
1	-	file not found
2	-	file too long
3	-	identifier expected
4	-	multiple definition of identifier
5	-	comma expected
6	-	left bracket expected
7	-	identifier not declared
8	-	data types do not fit together
9	-	division by zero
10	-	constant expected
11	-	operator expected
12	-	unexpected end of file
13	-	range operator '..' expected
14	-	right bracket ')' expected
15	-	'THEN' expected
16	-	'ELSE' expected
17	-	'ENDIF' expected
18	-	'THEN' without corresponding 'IF'
19	-	'ELSE' without corresponding 'IF'
20	-	'ENDIF' without corresponding 'IF'
21	-	assignment operator '=' expected
22	-	wrong format for integer number
23	-	wrong format for real number
24	-	formula too complicated
25	-	error in arithmetic expression
26	-	internal compiler error
27	-	identifier not valid
28	-	unknown type identifier
29	-	wrong input sign
30	-	stack overflow of parser

Format	Name	Description
31	-	syntax error
32	-	available memory exceeded
33	-	index or index set not allowed
34	-	error during dynamic storage allocation
35	-	wrong number of indices
36	-	wrong number of arguments
43	-	number of variables different from declaration
44	-	number of functions different from declaration
45	-	END - sign not allowed
46	-	Fortran code exceeds line
47	-	**: domain error
48	-	bad input format
49	-	length of working array IWA too small
50	-	length of working array WA too small
51	-	ATANH: domain error
52	-	LOG: domain error
53	-	SQRT: domain error
54	-	ASIN: domain error
55	-	ACOS: domain error
56	-	ACOSH: domain error
57	-	LABEL defined more than once
58	-	LABEL not found
59	-	wrong index expression
60	-	wrong call of the subroutine SYMINP
61	-	wrong call of the subroutine SYMPRP
62	-	compilation of the source file in GRAD-mode
63	-	interpolation values not in right order
64	-	not enough space for interpolation functions in subroutine REVCDE
65	-	length of working array IWA in subroutine SYMFOR too small
66	-	not enough interpolation values