NLPQL: A FORTRAN SUBROUTINE SOLVING CONSTRAINED NONLINEAR PROGRAMMING PROBLEMS

K. SCHITTKOWSKI

Institut für Informatik, Universität Stuttgart, Azenbergstrasse 12, D-7000 Stuttgart 1, West Germany

Abstract

NLPQL is a FORTRAN implementation of a sequential quadratic programming method for solving nonlinearly constrained optimization problems with differentiable objective and constraint functions. At each iteration, the search direction is the solution of a quadratic programming subproblem. This paper discusses the organization of NLPQL, including the formulation of the subproblem and the information that must be provided by a user. A summary is given of the performance of different algorithmic options of NLPQL on a collection of test problems (115 hand-selected or application problems, 320 randomly generated problems). The performance of NLPQL is compared with that of some other available codes.

Keywords and phrases

Nonlinear programming, sequential quadratic programming method, numerical implementation, test results.

1. Introduction

The code NLPQL was designed to solve the constrained nonlinear programming problem

$$x \in \mathbb{R}^n: \quad \begin{aligned} &\min\ f(x) \\ &g_j(x) = 0, \quad j = 1, \ldots, m_e, \\ &g_j(x) \geqslant 0, \quad j = m_e + 1, \ldots, m, \\ &x_\ell \leqslant x \leqslant x_u. \end{aligned} \tag{1}$$

The optimization method generates a sequence of quadratic programming subproblems which are to be solved successively. The algorithm is therefore known as the sequential quadratic programming (SQP) method. The theoretical details and some convergence results are found in Schittkowski [28]. Its domain of application is determined by the following assumptions:

(a) The problem is smooth: The problem functions are continuously differentiable on the set $E = \{x \in \mathbb{R}^n : x_\ell \leqslant x \leqslant x_u\}$. Note that the functions f and g_j, $j = 1, \ldots, m$, need to be defined only on the set E, since the iterates computed by the algorithm will never violate the lower and upper bounds.

(b) The problem is small: The problem size depends on hardware facilities, e.g. storage capacity, and on the capability of the quadratic programming routine to solve large problems. NLPQL was tested extensively on problems with up to 100 variables.

A rough outline of the method is given in sect. 2, to convey the underlying mathematical ideas and the usage of program modules which could be modified or replaced by a user. The organization of the program package is outlined in sect. 3, together with some implementation details. To facilitate the solution of a nonlinear programming problem of the form (1), easy-to-use versions are supplied either in the form of a main program or a subroutine. The standard problem-adaptable subroutine NLPQL1 contains additional features to alter default parameters or to fit the code into an existing system, e.g. by reverse communication. The program has been tested on about 700 test problems, in particular in the framework of the comparative studies of Hock and Schittkowski [15] and Schittkowski [26]. The test problems are either randomly generated with predetermined solution characteristics or gathered from the literature, cf. Hock and Schittkowski [15] and Schittkowski [26,30]. A few numerical results are presented in sect. 4 to show the dependence of the performance on the choice of some program modules. Moreover, the efficiency and reliability of NLPQL are compared with those of some other available nonlinear programming codes. The results are found in sect. 5.

The detailed usage of NLPQL is described in the user's guide (see Schittkowski [29]). The code is distributed upon request and has been used to solve practical nonlinear programming problems in many engineering and natural science areas. Most of the application problems come from mechanical engineering, particularly from structural optimization.

2. The sequential quadratic programming algorithm

Sequential quadratic programming methods for nonlinearly constrained optimization were developed mainly by Han [12,13] and Powell [20,21], based on the initial work of Wilson [31]. The principal idea is the formulation of a specific quadratic programming subproblem. Let x_k be a current iterate, v_k an approximation of the

optimal Lagrange multipliers, and B_k a positive definite approximation of the Hessian matrix of the Lagrangian function

$$L(x,u) := f(x) - \sum_{j=1}^{m'} u_j g_j(x) , \tag{2}$$

where $x \in \mathbb{R}^n$, $u = (u_1, \ldots, u_{m'})^T \in \mathbb{R}^{m'}$. Here we set $m' := m + 2n$, and define the bound constraints "$x_\ell \leqslant x \leqslant x_u$" by some functions $g_{m+1}(x), \ldots, g_{m'}(x)$ to simplify the notation. More precisely, we let

$$g_j(x) := x^{(j-m)} - x_\ell^{(j-m)}, \qquad j = m+1, \ldots, m+n ,$$

$$g_j(x) := x_u^{(j-m-n)} - x^{(j-m-n)}, \qquad j = m+n+1, \ldots, m' ,$$

where the right-hand sides include the components of x, x_ℓ, and x_u, respectively. By linearizing the nonlinear constraints of (1) and minimizing a quadratic approximation of the Lagrangian function (2), we obtain a subproblem of the form

$$\begin{aligned} & \min \tfrac{1}{2} d^T B_k d + \nabla f(x_k)^T d \\ d \in \mathbb{R}^n : \quad & \nabla g_j(x_k)^T d + g_j(x_k) = 0 , \quad j = 1, \ldots, m_e , \\ & \nabla g_j(x_k)^T d + g_j(x_k) \geqslant 0 , \quad j = m_e+1, \ldots, m , \\ & x_\ell - x_k \leqslant d \leqslant x_u - x_k . \end{aligned} \tag{3}$$

Let d_k be the solution of (3) and u_k the corresponding vector of Lagrange multipliers of the quadratic programming problem. Then a new iterate is determined by

$$x_{k+1} := x_k + \alpha_k d_k , \tag{4}$$

where α_k is a line search or steplength parameter. α_k is designed to produce a sufficient decrease of a merit function

$$\phi_k(\alpha) := \psi_{r_k} \left(\begin{pmatrix} x_k \\ v_k \end{pmatrix} + \alpha \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} \right) . \tag{5}$$

Since the line search may depend on the approximation v_k of the optimal Lagrange multipliers of (1), we update v_k simultaneously by

$$v_{k+1} := v_k + \alpha_k (u_k - v_k) . \tag{6}$$

In (5), r_k is a vector of penalty parameters and controls the degree of penalizing the objective or Lagrangian function when leaving the feasible region. Possible merit functions are the L_1-exact penalty function

$$\psi_r(x, v) := f(x) + \sum_{j=1}^{m_e} r_j |g_j(x)| + \sum_{j=m_e+1}^{m'} r_j |\min(0, g_j(x))| \tag{7}$$

used by Han [13] (see also Powell [20]), and the augmented Lagrangian function

$$\psi_r(x, v) := f(x) - \sum_{j=1}^{m_e} (v_j g_j(x) - \tfrac{1}{2} r_j g_j(x)^2)$$

$$- \sum_{j=m_e+1}^{m'} \begin{cases} (v_j g_j(x) - \tfrac{1}{2} r_j g_j(x)^2), & \text{if } g_j(x) \leqslant v_j/r_j , \\ \tfrac{1}{2} v_j^2 / r_j , & \text{otherwise} , \end{cases} \tag{8}$$

proposed by Schittkowski [27,28]. The penalty parameter r_k is updated by a suitable rule to guarantee a descent direction d_k with respect to the chosen merit function.

However, we can not always implement the quadratic programming subproblem (3) as it stands. It is possible that the feasible region of (3) will be empty although the original problem (1) is solvable. The second drawback is the recalculation of gradients of all constraints at each iteration, although some of them might be inactive at an optimal solution, i.e. locally redundant. To avoid both disadvantages, an additional variable δ and an active set strategy are introduced, leading to the modified subproblem

$$\begin{aligned} & \min \tfrac{1}{2} d^T B_k d + f(x_k)^T d + \tfrac{1}{2} \rho_k \delta^2 \\ d \in \mathbb{R}^n, \; & \nabla g_j(x_k)^T d + (1-\delta) g_j(x_k) \left\{ \begin{matrix} = \\ \geqslant \end{matrix} \right\} 0 , \quad j \in J_k , \\ \delta \in \mathbb{R} : \; & \nabla g_j(x_{k(j)})^T d + g_j(x_k) \geqslant 0 , \quad j \in K_k , \\ & x_\ell - x_k \leqslant d \leqslant x_u - x_k , \\ & 0 \leqslant \delta \leqslant 1 , \end{aligned} \tag{9}$$

where

$$J_k := \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m,\ g_j(x_k) \leq \epsilon \text{ or } v_j^{(k)} > 0\},$$

$$K_k := \{1, \ldots, m\} \setminus J_k .$$

Here we have $v_k = (v_1^{(k)}, \ldots, v_{m'}^{(k)})^T$ and ϵ is a user-provided tolerance. The index "$k(j)$" indicates gradients which have been calculated in previous iterations. The term ρ_k is an additional penalty parameter designed to reduce the influence of δ on a solution of (9). It is easy to see that the point $d_0 = 0, \delta_0 = 1$ satisfies the constraints of (9) and can also be used as a feasible starting point for a quadratic programming algorithm.

As an alternative to (9), a linear least-squares subproblem can be formulated by exploiting the factorization

$$B_k = L_k D_k L_k^T \tag{10}$$

with a lower triangular matrix L_k and a diagonal matrix D_k. The resulting linear least-squares problem then contains a triangular matrix in the objective function and is easily transformed into a least-distance problem, for which efficient algorithms are available. The LDL factors of B_k can be updated with about the same computational effort as the original matrix B_k, cf. for example Gill et al. [8].

The matrix B_k is to be a positive definite approximation for the Hessian matrix of the Lagrangian function (2). B_k can be updated by standard quasi-Newton techniques from unconstrained optimization. The BFGS formula is certainly the most popular one, and is implemented in NLPQL together with a modification proposed by Powell [20] to guarantee positive definite matrices.

The algorithm contains some additional features to overcome certain error situations and is completely described in Schittkowski [28]. Under some mild assumptions, it can be shown that the algorithm converges globally, i.e. starting from an arbitrary initial point, at least one accumulation point of the iterates will satisfy the Kuhn–Tucker optimality conditions. This result was proved for the augmented Lagrangian merit function (8). But for the L_1-merit function (7), a similar result can be achieved only under more stringent assumptions, e.g. that the penalty parameters are constant and sufficiently large. If the steplength is one in the neighbourhood of an optimal solution, then the algorithm is identical with the method investigated by Han [12] and by Powell [21]. Under different assumptions, they proved local superlinear convergence of the SQP method, which provides a theoretical justification for the fast final convergence speed we observe in practice.

3. Program organization and implementation details

The sequential quadratic programming algorithm described above has been implemented by the author and is distributed upon request. The program package does not contain a subroutine for solving the quadratic programming, or optionally linear least-squares, subproblem. The intention is that a user should utilize a library program or should obtain any available program, e.g. the quadratic programming codes QPSOL by Gill et al. [10], ZQPCVX by Powell [23], CONQUA and START by Kribbe [16], or the linear least-squares codes NNLS by Lawson and Hanson [18] and LCLSQ by Crane et al. [3].

The organization of the program package is explained by fig. 1, which also shows the program modules which have to be supplied by the user or which can be

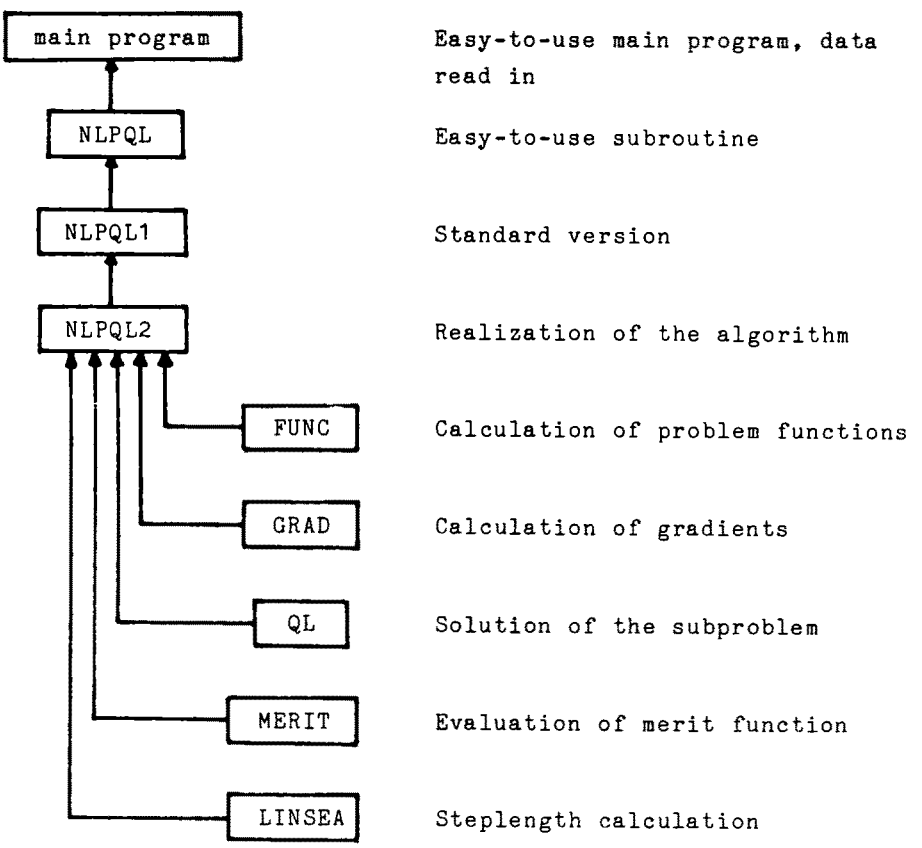

Fig. 1. Organization of the program package.

exchanged either to improve the submitted version or to test a specific module. To become familiar with the code, it is recommended that one first executes the easy-to-

use version, either in the form of a main program or subroutine. Only the problem dimension, number of constraints, desired output information, the bounds x_ℓ and x_u, a starting value x_0, and a subroutine evaluating the problem functions have to be supplied by a user. All other decisions are predetermined by default values. If, however, this approach fails and some algorithmic parameters are to be changed, or if the user wants to have more influence on the solution process, the standard subroutine NLPQL1 can be executed. Here, additional input information about the problem can be provided, an automatic scaling procedure applied, or reverse communication performed. Subroutine FUNC has to be defined by the user to calculate the problem functions. If analytical differentiation is to be performed, subroutine GRAD must also be implemented by the user. As mentioned before, any quadratic programming or linear least-squares program can be used to solve the subproblem defined in sect. 2. To link this specific code with the nonlinear programming algorithm, it must be implemented in a special way, i.e. within a subroutine with the name QL and a fixed, predetermined calling sequence. The set of active constraints and the function or gradient values of the merit function (5) are calculated by a subroutine with the name MERIT. The program package contains a realization of the augmented Lagrangian function (8), but it could be replaced by any other merit function, e.g. the L_1-exact penalty function (7). Finally a user may influence the line search procedure, which is performed in a subroutine with the name LINSEA. The program package offers a simple Armijo-type bisection method combined with quadratic interpolation, but this scheme could be replaced by a more sophisticated sub-algorithm, e.g. by GETPTC of Gill et al. [9]. Detailed information about the usage of the nonlinear programming code and its modules is given in Schittkowski [29].

As mentioned before, the subroutine NLPQL1 allows the alteration of default values or some internal algorithmic decisions, and to adapt the solution process to a specific situation. The most important features are the following ones:

1. Alternative subproblems: A logical variable must be set by the user in order to indicate whether a quadratice programming or, alternatively, a linear least-squares problem is to be formulated. When using Powell's program ZQPCVX for example, one could decide whether the initial Cholesky factorization of the Hessian approximation is to be performed within ZQPCVX or the outer nonlinear programming algorithm.

2. Expanded quadratic programming problem: In normal execution, NLPQL formulates the quadratic programming subproblem (3) combined with an active set strategy to avoid numerical inaccuracies induced by the additional variable δ. This additional variable is introduced only if the corresponding algorithm used to solve that subproblem reports an error message. Then one obtains a subproblem of the form (9). Alternatively, this subproblem can be formulated in every iteration step of NLPQL1, so that the specific algorithm for solving it is always provided with a feasible starting point.

3. Scaling: Scaling is among the most difficult problems in practical optimization. Roughly speaking, one tries to achieve a model formulation, such that a small fixed alteration of any variable induces an alteration of the problem functions of the same order of magnitude. A generally applicable scaling method is not available, since initially the nonlinear programming algorithm possesses information about the behaviour of the problem only in a neighbourhood of the starting point. Thus, the best recommendation for a user is to define scaling parameters that depend on the particular practical problem. Nevertheless, a very careful scaling procedure is included in NLPQL1. If the objective function value or a constraint violation at the starting point is greater than a default value (1000), then the corresponding functions are scaled by the factors $1/\sqrt{|f(x_0)|}$ or $1/\sqrt{|g_j(x_0)|}$, respectively. Of course, the default value can be changed by a user, and when defining any negative value, e.g. -1, he (she) is allowed to predetermine his (her) own scaling parameters in a working array.

4. Reverse communication: Nonlinear programming codes are often applied as auxiliary routines in complex systems, e.g. as part of an optimal control or finite element algorithm. In these cases, it might be helpful to use reverse communication, the most flexible way to solve an optimization problem. Only one iteration step will be performed by NLPQL1. Then the subroutine returns to the main program of the user, where new function and gradient values have to be evaluated. A subsequent call of NLPQL1 continues the iteration.

5. Additional problem information: Initially, the approximation matrix for the Hessian of the Lagrangian function is set to the identity matrix and the initial estimates for the multipliers are set to zero. Alternatively, a user could provide the program with his (her) own guesses, to exploit known information about the problem structure.

6. Restart in error cases: If requested by a user, NLPQL1 performs automatic restarts in error situations (see remark 8 below). Proceeding from the last computed iterate, the quasi-Newton matrix, the multiplier estimates, and the penalty parameters are all set to their initial values and an attempt is made to solve the problem again, now with a lower limit on the allowed number of iterations.

7. Output facilities: A user is allowed to suppress all output. Alternatively, a final convergence analysis or additional output for each iteration step can be produced. Some further output from the program modules mentioned above can be printed upon request.

8. Error conditions: The user will be informed about the reason for termination of the algorithm. If the optimality conditions could not be satisfied to within some user-provided tolerance, then the code encountered a certain error situation, which is reported. The following errors occur most frequently:

- The algorithm terminates because the user-provided maximum number of iterations was attained.

- The line search algorithm stopped because the user-provided maximum number of sub-iterations was exceeded. This situation occurs, for example, if the iterates are close to a solution but the optimality conditions can not be satisfied due to round-off errors.
- The search direction d_k is close to zero, but the current iterate is still infeasible. The message indicates badly scaled problem functions.

The program requires core storage for approximately $n^2 + mn + 28n + 9m$ real variables, plus whatever additional storage is needed to solve the quadratic programming subproblem. The problem size is therefore limited by the core size and the capability of the subproblem algorithm to solve large problems. The program has been implemented in FORTRAN and tested by the author on a Telefunken TR440 at the University of Würzburg, on an IBM 370/168 at the Stanford University, and on a VAX 11/780 at the University of Stuttgart in single and double precision arithmetic. Moreover, the code has been run by users on many other mainframe machines.

Besides NLPQL, some other nonlinear programming codes realizing a sequential quadratic programming algorithm are available upon request. The first implementation of an SQP method is Powell's program VF02AD [20], which was distributed by the Harwell Subroutine Library. The algorithm uses the L_1-penalty function (7) to determine a steplength, and Fletcher's [6] quadratic programming routine VE02AD. A very similar implementation was performed by Crane et al. [4]. The resulting program is called VMCON. Based on the observation that the L_1-merit function could prevent superlinear convergence or even cycle, a watchdog technique was proposed by Chamberlain et al. [2]. The resulting program VMCWD by Powell [22] was compared numerically with NLPQL (see Powell [24] for details). For a class of highly nonlinear test problems, the augmented Lagrangian merit function seems to be preferable, but this function could induce some numerical instabilities when the problem is degenerate, i.e. when the gradients of active constraints are linearly dependent. Gill et al., [11] implemented an SQP method (SOL/NPSOL) which also uses the augmented Lagrangian merit function, which is now included in the NAG Library.

4. Testing different program modules

We now give the results obtained by interchanging the program modules outlined in sect. 3. The intention is to form an impression about the numerical sensitivity of the nonlinear programming method, if one of its sub-algorithms is altered. The resulting different versions of NLPQL1 are identified by the parameter NC as shown in table 1, where the following abbreviations are used:

QL :	Subroutine for calling a quadratic programming or linear least-squares algorithm;
QL = QPSOL :	A quadratic programming subproblem is formulated and solved by algorithm QPSOL by Gill et al. [10];

QL = NNLS : A linear least-squares subproblem is formulated, transformed into a least-squares problem with lower bounds on the variables only, and solved by subroutine NNLS by Lawson and Hansen [18];

LINSEA : Subroutine for performing a one-dimensional minimization, i.e. for calculating a steplength α_k;

LINSEA = QI : Simple quadratic interpolation is used, combined with an Armijo-type stopping criterion, cf. Powell [20] or Schittkowski [27,28];

LINSEA = GETPTC : This safeguarded cubic interpolation algorithm was developed by Gill et al. [9] and requires simultaneous function and gradient evaluations during the line search;

MERIT : Subroutine MERIT evaluates the merit or line search function, the active set, and the penalty parameters;

MERIT = L1 : An L_1-exact penalty function is formulated, cf. Han [13] or Powell [20]. All constraints are contained in the active set and the penalty parameters are defined following Powell [20];

MERIT = L2 : An augmented Lagrangian function is defined, where the Lagrangian function is penalized in the L_2-norm, cf. Schittkowski [27,28]. This function is continuously differentiable, and an active set strategy is performed;

GRAD : This subroutine is required to compute the gradients of the problem functions;

GRAD = A : Analytical derivatives are provided by the test frame;

GRAD = N : The gradients are approximated numerically by scaled forward differences.

For the numerical test runs, the test problems of Hock and Schittkowski [14] were used. Only some mean values will be presented here to give an impression of the performance of NLPQL1 in the situations of concern. First a decision must be made as to whether a test run was successful or not. To avoid unfair comparisons if different local solutions of a problem are obtained, the few corresponding problems have been dropped from the evaluation of the test results. Then a test run is called successful if the error in the objective function is less than 10^{-3} and if the constraint violation is less than 10^{-4}. The numerical tests were performed on an IBM 370/168 at Stanford University in double precision arithmetic using the WATFIV compiler. The following notation is needed in table 2 to explain the results:

Table 1

NLPQL1 versions

NC	QL	LINSEA	MERIT	GRAD
1	QPSOL	QI	L2	A
2	QPSOL	QI	L1	A
3	QPSOL	GETPTC	L2	A
4	QPSOL	QI	L2	N
5	NNLS	QI	L2	A

Table 2

Average test results for different versions of NLPQL1

NC	PNS	NF	NG	NDF	NDG	ITER	DGX	DFX
1	10.4	14.6	64.6	11.9	28.2	11.9	0.37E-10	0.22E-9
2	7.8	13.8	58.2	11.6	51.5	11.6	0.20E-10	0.15E-9
3	10.4	13.9	59.5	13.9	33.0	11.0	0.20E-10	0.16E-9
4	7.8	83.5	313.0	0.0	0.0	12.3	0.96E-10	0.52E-9
5	25.3	(12.3)	(36.6)	(9.7)	(18.6)	(9.7)	(0.13E-8)	(0.12E-7)

NC : Version number of NLPQL1, cf. table 1;

PNS : Percentage of non-successful solutions;

NF : Average number of objective function evaluations;

NG : Average number of constraint function evaluations, each constraint counted;

NDF : Average number of gradient evaluations of the objective function;

NDG : Average number of gradient evaluations of the constraint functions, each constraint counted;

ITER : Average number of iterations (identical with NDF, if analytical derivatives and no gradient evaluations for the line search are used);

DGX : Average violation of constraints corresponding to the successful test runs (geometric mean);

DFX : Average error in objective function corresponding to the successful test runs (geometric mean).

All mean values are evaluated only for successful test runs. There are no drastic differences in the performance of all these versions of NLPQL1. The slight differences can be summarized as follows.

NC = 1 versus NC = 2: The outstanding difference is that many gradient evaluations of the constraint functions can be saved by the active set strategy. When using the L_1-penalty function, two more problems (TP74/75) could be solved successfully, since the approximation of a point where the constraint qualification of (1) is not satisfied could be prevented.

NC = 1 versus NC = 3: There is nearly no difference in the performance of both versions. The number of gradient evaluations for NC = 3 is only slightly greater than for NC = 1, showing that in most iterations the steplength one is satisfactory.

NC = 1 versus NC = 4: The use of numerical differentiation does not seem to have a significant effect on the number of successful runs, the number of iterations, or the accuracy of the solutions.

NC = 1 versus NC = 5: Obviously, the least-squares version is less reliable than the version based on quadratic programming subproblems, because of the numerically unstable transformation of (3) into the form required by NNLS. This is also verified by a lower final accuracy of this version. The reduced number of function and gradient evaluations is due to the fact that many of the more complicated, higher dimensional test problems could not be solved. An evaluation of the mean values based on those test runs which were successful for both versions would not indicate any significant differences. Therefore, the performance figures of the least-squares versions we set between brackets and they should be used very carefully when comparing them with those of the other versions.

The preceding results show, in particular, the flexibility of NLPQL1, since all implemented versions represent different mathematical ways to solve the corresponding subproblem.

5. Some comparative test results

NLPQL was tested in the framework of a comparative study of optimization codes, cf. Schittkowski [26]. Using the same randomly generated test problems and the same evaluation system for the results, the performance of NLPQL was compared with that of 26 other available nonlinear programming codes. 370 additional test runs have been performed to obtain these results, on a Telefunken TR440 at the University

of Würzburg in single precision arithmetic. A numerical comparison of NLPQL with the optimization codes based on the test problems of Hock and Schittkowski [14] is also possible, cf. Hock and Schittkowski [15].

To give at least a rough impression of the numerical performance of NLPQL, a few results are compared with those of seven other optimization programs which are frequently used in practical applications. The programs are typical realizations of the underlying mathematical method and represent the algorithmic progress in nonlinear programming during the last 15 years. Their names, authors, and mathematical methods are found in table 3. More detailed information about these and 19 other

Table 3

Optimization programs

Code	Author	Method
SUMT	Fiacco, McCormick [5]	Penalty method
NLP	Rufer [25]	
VF01A	Fletcher [7]	Multiplier method
LPNLP	Pierre, Lowe [19]	
GRGA	Abadie [1]	Generalized reduced gradient method
GRG2	Lasdon, Waren [17]	
VF02AD	Powell [20]	Sequential quadratic programming method
NLPQL	Schittkowski [28]	

available optimization codes is given in Schittkowski [26]. Table 4 contains some average efficiency and reliability scores obtained by a sequence of 240 test runs for each code. In this case, the test problems are randomly generated with predetermined solutions (see Schittkowski [26] for details). The following abbreviations are used in table 4 to characterize the numerical performance.

ET : Average execution time in seconds;

NF : Average number of objective function calls;

NG : Average number of constraint function calls, each single constraint counted;

NDF : Average number of gradient calls of the objective function;

NDG : Average number of gradient calls of the constraint functions, each single constraint counted;

PNS : Percentage of non-successful solutions and of failures (overflow, exceeding calculation times, etc.).

Table 4

Numerical results

Code	ET	NF	NG	NDF	NDG	PNS
SUMT	270.1	2335	24 046	99	1053	77.8
NLP	88.1	1043	8635	111	957	28.5
VF01A	42.2	158	1595	158	603	29.3
LPNLP	57.8	252	2518	101	1014	30.5
GRGA	37.7	204	2946	67	378	13.4
GRG2	52.6	297	3368	38	423	10.4
VF02AD	31.6	16	179	16	179	8.3
NLPQL	14.1	18	181	16	64	3.3

Obviously, the sequential quadratic programming codes are the most efficient ones, followed by the generalized reduced gradient, multiplier, and penalty methods. The programs NLPQL and VF02AD use about the same number of function evaluations and iterations, since both codes are based on the same mathematical idea. However,NLPQL saves many gradient evaluations of the constraints due to the active set strategy, leading also to a reduction of calculation time. A further reduction is obtained by using the more efficient subroutine QPSOL by Gill et al. [10], which was implemented in NLPQL for solving the quadratic programming subproblem.

References

[1] J. Abadie, Méthode du gradient reduit generalisé: Le code GRGA, Note HI 1756/00, Electricite de France, Paris (1975).

[2] R.M. Chamberlain, C. Lemarechal, H.C. Pedersen and M.J.D. Powell, The watchdog technique for forcing convergence in algorithms for constrained minimization, Mathematical Programming Studies 16(1982)1.

[3] R.L. Crane, B.S. Garbow, K.E. Hillstrom and M. Minkoff, LCLSQ: An implementation of an algorithm for linearly constrained linear least squares problems, Report ANL-80-116, Argonne National Laboratory, Argonne, Illinois (1980).

[4] R.L. Crane, K.E. Hillstrom and M. Minkoff, Solution of the general nonlinear programming problem with subroutine VMCON, Report ANL-80-64, Argonne National Laboratory, Argonne, Illinois (1980).

[5] A.V. Fiacco and G.P. McCormick, *Nonlinear Sequential Unconstrained Minimization Techniques* (Wiley, New York, 1968).

[6] R. Fletcher, A FORTRAN program for general quadratic programming, Report No. R6370, AERE, Harwell, Berkshire (1970).

[7] R. Fletcher, An ideal penalty function for constrained optimization, in: *Nonlinear Programming 2*, ed. O.L. Mangasarian, R.R. Meyer and S.M. Robinson (Academic Press, New York, 1975).

[8] P.E. Gill, W. Murray and M.A. Saunders, Methods for computing and modifying the LDV factors of a matrix, Mathematics of Computation 29(1975)1051.

[9] P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright, Two steplength algorithms for numerical optimization, Report SOL 79-25, Dept. of Operations Research, Stanford University, Stanford (1979).

[10] P.E. Gill, W. Murray, M.A. Saunders and M. Wright, User's guide for SOL/QPSOL: A FORTRAN package for quadratic programming, Report SOL 82-7, Dept. of Operations Research, Stanford University (1982).

[11] P.E. Gill, W. Murray, M.A. Saunders and M. Wright, User's guide for SOL/NPSOL: A FORTRAN package for nonlinear programming, Report SOL 83-12, Department of Operations Research, Stanford University (1983).

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

[13] S.-P. Han, A globally convergent method for nonlinear programming, J. of Optimization Theory and Applications 22(1977)297.

[14] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems*, Vol. 187 (Springer-Verlag, Berlin – Heidelberg – New York, 1981).

[15] W. Hock and K. Schittkowski, A comparative performance evaluation of 27 nonlinear programming codes, Computing 30(1983)335.

[16] W. Kribbe, Documentation of the FORTRAN-subroutines for quadratic programming CONQUA and START, Report 8231/1, Econometric Institute, Erasmus University, Rotterdam (1982).

[17] L.S. Lasdon and A.D. Waren, Generalized reduced gradient software for linearly and nonlinearly constrained problems, in: *Design and Implementation of Optimization Software*, ed. H.J. Greenberg (Sijthoff and Noordhoff, Alphen aan den Rijn (1978).

[18] C.L. Lawson and R.J. Hanson, *Solving Least Squares Problems* (Prentice Hall, Englewood Cliffs, New Jersey, 1974).

[19] D.A. Pierre and M.J. Lowe, *Mathematical Programming via Augmented Lagrangians* (Addison – Wesley, Reading, Massachusetts, 1975).

[20] M.J.D. Powell, A fast algorithm for nonlinearly constrained optimization calculations, in: *Numerical Analysis*, ed. G.A. Watson, *Lecture Notes in Mathematics*, Vol. 630 (Springer-Verlag, Berlin – Heidelberg – New York, 1978).

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

[22] M.J.D. Powell, VMCWD: A FORTRAN subroutine for constrained optimization, Report DAMTP 1982/NA4, University of Cambridge, Cambridge (1982).

[23] M.J.D. Powell, ZQPCVX: A FORTRAN subroutine for convex quadratic programming, Report DAMTP 1983/NA17, University of Cambridge, Cambridge (1983).

[24] M.J.D. Powell, The performance of two subroutines for constrained optimization on some difficult test problems, Report DAMTP 1984/NA6, University of Cambridge, Cambridge (1984).

[25] D. Rufer, User's guide for NLP – A subroutine package to solve nonlinear optimization problems, Report No. 78-07, Fachgruppe für Automatik, ETH Zürich (1978).

[26] K. Schittkowski, *Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems*, Vol. 183 (Springer-Verlag, Berlin – Heidelberg – New York, 1980).

[27] K. Schittkowski, The nonlinear programming method of Wilson, Han, and Powell with an augmented Lagrangian type line search function. Part 1: Convergence analysis, Numerische Mathematik 38(1981)83.

[28] K. Schittkowski, On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function, Mathematische Operationsforschung und Statistik, Ser. Optimization 14(1983)197.
[29] K. Schittkowski, User's guide for the nonlinear programming code NLPQL, Report, Institut für Informatik, Universität Stuttgart, FRG (1984).
[30] K. Schittkowski, Test examples for nonlinear programming codes, Report, Institut für Informatik, Universität Stuttgart, FRG (1984).
[31] R.B. Wilson, A simplicial algorithm for concave programming, Ph.D. Thesis, Graduate School of Business Administration, Harvard University, Boston (1963).