NLPQLB: A Fortran Implementation of an SQP Algorithm with Active Set Strategy for Solving Optimization Problems with a Very Large Number of Nonlinear Constraints

- User's Guide -

Address: Prof. K. Schittkowski
Department of Computer Science
University of Bayreuth
D - 95440 Bayreuth

Phone: (+49) 921 557750

Fax: +921 35557

E-mail: klaus.schittkowski@uni-bayreuth.de

Web: http://www.klaus-schittkowski.de

Date: November, 2010

Abstract

The Fortran subroutine NLPQLB solves smooth nonlinear programming problems with a large number of constraints, but a moderate number of variables. The underlying algorithm applies an active set method proceeding from a given bound m_w for the maximum number of expected active constraints. A quadratic programming subproblem is generated with m_w linear constraints, the so-called working set, which are internally exchanged from one iterate to the next. Only for active constraints, i.e., a certain subset of the working set, new gradient values must be computed. The line search takes the active constraints into account. In case of computational errors as for example caused by inaccurate function or gradient evaluations, a non-monotone line search is activated. Numerical results are included for some academic test problems, which show that nonlinear programs with up to 200,000,000 nonlinear constraints can be efficiently solved. The amazing observation is that despite of a large number of nearly dependent active constraints, the underlying SQP code converges very fast. The usage of the code is documented and illustrated by an example.

Keywords: SQP, sequential quadratic programming, nonlinear programming, large number of constraints, active set strategy, active constraints, non-monotone line search

1 Introduction

We consider the general optimization problem to minimize an objective function under nonlinear equality and inequality constraints,

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

where x is an n-dimensional parameter vector. It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on the whole $I\!R^n$. To simplify the notation, we omit the upper and lower bounds and get a problem of the form

$$x \in I\!R^n : \begin{array}{ll} \min & f(x) \\ & g_j(x) = 0 \ , \quad j = 1, \ldots, m_e \ , \\ & g_j(x) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ . \end{array} \tag{2}$$

We assume now that the nonlinear programming problem possesses a very large number of nonlinear inequality constraints on the one hand, but a much lower number of variables. A typical situation is the discretization of an infinite number of constraints, as indicated by the following case studies.

1. Semi-infinite optimization: Constraints must be satisfied for all $y \in Y$, where y is an additional variable and $Y \subset I\!R^r$,

$$x \in I\!R^n : \begin{array}{ll} \min & f(x) \\ & g(x, y) \geq 0 \quad \text{for all } y \in Y \ . \end{array} \tag{3}$$

Here we assume for simplicity that there is only one scalar restriction of inequality type. If we discretize the set Y, we get a standard nonlinear programming problem, but with a large number of constraints depending on the desired accuracy,

$$x \in I\!R^n : \begin{array}{ll} \min & f(x) \\ & g(x, y_j) \geq 0 \ , \quad j = 1, \ldots, m \ , \end{array} \tag{4}$$

where y_j is a discretization of Y, e.g., $y_j = \frac{j-1}{m-1}$ for $j = 1, \ldots, m$ in case of $Y = [0, 1]$.

2. Min-max optimization: We minimize the maximum of a function f depending now on two variables $x \in I\!R^n$ and $y \in I\!R^r$,

$$\min_{x \in X} \max_{y \in Y} \ f(x, y) \tag{5}$$

with suitable subsets $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^r$, respectively. (5) is easily transformed into an equivalent standard nonlinear programming problem of the form

$$x \in X : \begin{array}{l} \min t \\ f(x,y) \le t \quad \text{for all } y \in Y \ . \end{array} \tag{6}$$

Again, we get a semi-infinite optimization problem provided that Y is an infinite set.

3. L_∞-Approximation: The situation is similar to min-max optimization, but we want to minimize the maximum of absolute values of a given set of functions,

$$\min_{x \in \mathbb{R}^n} \max_{i=1,\ldots,r} |f_i(x)| \ . \tag{7}$$

Typically, the problem describes the approximation of a nonlinear functions by a simpler function, i.e., $f_i(x) = f(t_i) - p(t_i, x)$. In this case, $f(t)$ is a given function depending on a variable $t \in \mathbb{R}$ and $p(t,x)$ a member of a class of approximating functions, e.g., a polynomial in t with coefficients $x \in \mathbb{R}^n$. The problem is non-differentiable, but can be transformed into a smooth one assuming that all functions $f_i(x)$ are smooth,

$$x \in \mathbb{R}^n : \begin{array}{ll} \min & t \\ f_i(x) \le t & i = 1, \ldots, r \ , \\ f_i(x) \ge -t & i = 1, \ldots, r \ . \end{array} \tag{8}$$

4. Optimal control: The goal is to determine a control function $u(t)$ depending on a time variable t, which has to minimize a cost criterion subject to a state equation in form of a system of differential equations and additional restrictions on the state and control variables that are to be satisfied for all time values under consideration. If the control problem is discretized in a proper way, we get a nonlinear programming problem with a large number of constraints.

5. Mechanical structural optimization: Suppose we want to minimize the weight of a mechanical structure. Typical constraints in this case are bounds for allowable stresses or displacements, among very many other possible types of restrictions. After modeling the structure by a finite element technique, we will get a standard nonlinear programming problem. If certain restrictions are to be taken into account for each element of the structure, we will get optimization problems with a large number of constraints. Moreover a design engineer may wish to define different load cases, where part of the constraints is duplicated subject to some other data. In Kneppe [5] it is reported that the optimization of the frame of an airplane fuselage lead to an optimization problem with 187 variables and 92,829 nonlinear constraints.

The examples mentioned above motivate the necessity to develop special methods for problems with very many restrictions. The total number of constraints is so large that

either the linearized constraints cannot be stored in memory or slow down the solution process unnecessarily. Although we can expect that most of the constraints are redundant, we cannot predict a priori which constraints are the important ones, i.e., probably active at the optimal solution, and which not.

Sequential quadratic programming methods construct a sequence of quadratic programming subproblems by approximating the Lagrangian function

$$L(x,u) \doteq f(x) - \sum_{j=1}^{m} u_j g_j(x) \tag{9}$$

quadratically and by linearizing the constraints. The resulting quadratic programming subproblem

$$\begin{aligned} & \min \ \tfrac{1}{2} d^T B_k d + \bigtriangledown f(x_k)^T d \\ d \in I\!R^n, \delta \in I\!R: \ & \bigtriangledown g_j(x_k)^T d + g_j(x_k) = 0 \ , \ j = 1, \dots m_e \ , \\ & \bigtriangledown g_j(x_k)^T d + g_j(x_k) \geq 0 \ , \ j = m_e + 1, \dots m \end{aligned} \tag{10}$$

can be solved by any available *black-box* algorithm, at least in principle. Here, x_k denotes an actual iterate and B_k an estimate of the Hessian of the Lagrangian function (9) updated by the BFGS quasi-Newton method. However, if m is large, the Jacobian might become too big to be stored in the computer memory.

The basic idea is to proceed from a user-provided value m_w with

$$n \leq m_w \leq m$$

by which we estimate the maximum number of expected active constraints. Only quadratic programming subproblems with m_w linear constraints are created which require lower storage and allow faster numerical solution. Thus, one has to develop a strategy to decide, which constraint indices are added to a working set of size m_w

$$W \doteq \{j_1, \dots, j_{m_w}\} \subset \{1, \dots, m\}$$

and which ones have to leave the working set. It is recommended to keep as many constraints as possible in a working set, i.e., to keep m_w as large as possible, since also non-active constraint could contribute an important influence on the computation of search direction.

It is, however, possible, that too many constraints are violated at a starting point even if it is known that the optimal solution possesses only very few active constraints. To avoid an unnecessary blow-up of the working set, it is also possible to extend the given optimization problem by an additional artificial variable x_{n+1}, which, if chosen sufficiently large at start, decreases the number of active constraints. (1) or (2), respectively, is then

replaced by

$$x \in I\!R^{n+1} : \begin{array}{ll} \min f(x) + \rho x_{n+1} & \\ g_j(x) = 0 \,, & j = 1, \ldots, m_e \;\; , \\ g_j(x) + x_{n+1} \geq 0 \,, & j = m_e + 1, \ldots, m \;\; , \\ x_{n+1} \geq 0 \;\; . & \end{array} \tag{11}$$

However, this transformation does not make sense in cases where the original problem is transformed in a similar way as for example the min-max problem (6). Also, the choice of the penalty parameter ρ and the starting value for x_{n+1} is crucial. A too rapid decrease of x_{n+1} to zero must be prevented to avoid too many active constraints, which is difficult to achieve in general. But if adapted to a specific situation, the transformation works very well and can be extremely helpful.

There is another motivation for considering active sets. Since we want to solve problems with a large number of constraints, many of them are probably redundant. But in any case, we have to require the evaluation of gradients for all of them in the working set. Thus, an additional active set strategy is proposed with the aim to reduce the number of gradient evaluations, and to calculate gradients at a new iterate only for a certain subset of estimated active constraints. The underlying SQP algorithm is described in Schittkowski [10], and the presented active set approach for solving problems with a large number of constraints in Schittkowski [13].

Active set strategies are widely discussed in the nonlinear programming literature and have been implemented in most of the available codes. A computation study for linear constraints was even conducted in the 70's, see Lenard [6], and Google finds 267,000 hits for `active set strategy nonlinear programming`. It is out of the scope of this paper to give a review. Some of these strategies are quite complex and a typical example is the one included in the KNITRO package for large scale optimization, see Byrd, Gould, Nocedal, and Waltz [1], based on linear programming and equality constrained subproblems.

From the technical point of view, NLPQLB is implemented in form of a Fortran subroutine, where function and gradient values are passed through reverse communication, see Schittkowski [17]. NLPQLB calls the SQP code NLPQLP, see again [17], with exactly m_w constraints, where the constraints of the working set are changed from one iteration to the next.

The modified SQP-algorithm is described in Section 2 in detail. Since some heuristics are included which prevent a rigorous convergence analysis, at least the most important sufficient decrease property is available which shows that the algorithm is well-defined. Some numerical test results based on a few academic examples are found in Section 3, where the number of nonlinear constraints is very large, i.e., up to 200,000,000. More details of the software implementation and the usage of the code NLPQLB are presented in Section 4. Section 5 contains a simple example to become familiar with the software.

2 An Active-Set Sequential Quadratic Programming Method

Since we want to modify the sequential quadratic programming algorithm presented in Schittkowski [10, 11], we use the notation introduced there and omit details which are not essential to understand the basic idea. A typical dense SQP code takes all constraints into account, and requires real working arrays of length $O(n^2 + nm)$, see Schittkowski [17]. n is the number of variables and m the number of constraints of (1) without bounds. In particular, we need mn double precision real numbers to store the gradients of the constraint functions for the quadratic programming subproblem.

We assume now that n is of reasonable size, say below 100, but that m is very large compared to n, say 1,000,000 or even more. Then either the available memory is insufficient to store the total gradient matrix of size nm, or the large set of linear constraints in the subproblem slows down the quadratic programming solver. It is furthermore assumed that there are no sparsity patterns in the Jacobian matrix which could be exploited. Thus, we replace m by m_w, where m_w is a user-provided number depending on the available memory and the expected number of active constraints, and which satisfies

$$n \leq m_w \leq m \quad .$$

It is supposed that a double precision array of size nm_w can be addressed in memory. Moreover, it has to be guaranteed that the active-set algorithm is identical with a standard SQP method if $m = m_w$.

We want to formulate smaller quadratic programming subproblems with m_w linear constraints. If x_k denotes an iterate of the algorithm, v_k the corresponding multiplier estimate and B_k a positive definite estimate of the Hessian of the Lagrangian function (9), we solve quadratic programs of the form

$$\begin{aligned} & \min \ \tfrac{1}{2} d^T B_k d + \bigtriangledown f(x_k)^T d \ + \tfrac{1}{2}\sigma_k^2 \delta^2 \quad , \\ d \in I\!R^n, \delta \in I\!R: \ & \bigtriangledown g_j(x_k)^T d + (1-\delta) g_j(x_k) \ \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} \ 0 \ , \ j \in J_k^\star \quad , \\ & \bigtriangledown g_j(x_{j(k)})^T d + g_j(x_k) \geq 0 \ , \ j \in \overline{K}_k^\star \quad . \end{aligned} \tag{12}$$

An additional variable δ is introduced to prevent infeasible linear constraints, see Schittkowski [10] for details. To get a descent direction in this case, it may happen that another internal loop must be entered with an increasing penalty term for the additional variable, see Schittkowski [10] for details. Note that the matrix B_k is positive definite, so that the solution of (12) is always unique.

The index set $J_k^\star$ is called the set of active constraints and is defined by

$$J_k^\star \doteq \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, \ g_j(x_k) < \epsilon \text{ or } v_j^{(k)} > 0\} \quad . \tag{13}$$

It is assumed that $|J_k^\star| \leq m_w$, i.e., that all active constraints are part of the working set

$$W_k \doteq J_k^\star \cup \overline{K}_k^\star \tag{14}$$

with m_w elements. The working set contains the active constraints plus a certain subset of the non-active ones, $\overline{K}_k^\star \subset K_k^\star$, defined by

$$K_k^\star := \{1, \ldots, m\} \setminus J_k^\star \quad . \tag{15}$$

$v_k = (v_1^{(k)}, \ldots, v_m^{(k)})^T$ is the actual multiplier estimate and ϵ a user provided error tolerance. The indices $j(k)$ in (12) denote previously computed gradients of constraints. Their definition will become clear when investigating the algorithm in more detail. The idea is to recalculate only gradients of active constraints and to fill the remaining rows of the constraint matrix with previously computed ones.

We have to assume that there are not more than m_w active constraints throughout the algorithm. But we do not support the idea to include some kind of automatized *phase I* procedure to project an iterate back to the feasible region whenever this assumption is violated. We will have some safeguards in the line search algorithm to prevent this situation. If, for example at a starting point, more than m_w constraints are active, it is preferred to stop the algorithm and to leave it to the user either to change the starting point or to establish an outer constraint restoration procedure depending on the problem structure.

After solving the quadratic programming subproblem (12) we get a search direction d_k and a corresponding multiplier vector u_k. The new iterate is obtained by

$$x_{k+1} \doteq x_k + \alpha_k d_k \quad , \quad v_{k+1} \doteq v_k + \alpha_k (u_k - v_k) \tag{16}$$

for approximating the optimal solution $x^\star \in I\!R^n$ of (2) and the corresponding optimal multiplier vector $u^\star \in I\!R^m$. The steplength parameter α_k is the result of an additional line search sub-algorithm, by which we want to achieve a sufficient decrease of an augmented Lagrangian merit function

$$\psi_r(x, v) \doteq f(x) - \sum_{j \in J(x,v)} (v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2) - \frac{1}{2} \sum_{j \in K(x,v)} v_j^2 / r_j \quad . \tag{17}$$

The index sets $J(x, v)$ and $K(x, v)$ are defined by

$$\begin{aligned} J(x, v) &\doteq \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, g_j(x) \leq v_j / r_j\} \quad , \\ K(x, v) &\doteq \{1, \ldots, m\} \setminus J(x, v) \quad , \end{aligned} \tag{18}$$

see Schittkowski [10]. The corresponding penalty parameters $r_k \doteq (r_1^k, \ldots, r_m^k)^T$ that control the degree of constraint violation, must carefully be chosen to guarantee a sufficient descent direction of the merit function

$$\phi_{r_k}(\alpha_k) \leq \phi_{r_k}(0) + \alpha_k \mu \phi'_{r_k}(0) \quad , \tag{19}$$

see Schittkowski [10], Ortega and Rheinboldt [8], or Wolfe [19] in a more general setting, where

$$\phi_{r_k}(\alpha) \doteq \psi_{r_k}\left(\begin{pmatrix} x_k \\ v_k \end{pmatrix} + \alpha \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix}\right) \tag{20}$$

and

$$\phi'_{r_k}(0) = \bigtriangledown \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} \quad . \tag{21}$$

An additional requirement is that at each intermediate step of the line search procedure at most m_w constraints are active. If this condition is violated, the steplength is further reduced until satisfying this condition. From the definition of our index sets, we have

$$J_k^\star \supset J_k \doteq J(x_k, v_k) \quad . \tag{22}$$

The starting point x_0 is crucial from the viewpoint of numerical efficiency and must be predetermined by the user. It has to satisfy the assumption that not more than m_w constraints are active, i.e., that $J_0 \subset W_0$. The remaining indices of W_0 are to be set in a suitable way and must not overlap with the active ones. Also W_0 must be provided by the user to have the possibility to exploit pre-existing knowhow about the position of the optimal solution and its active constraints.

For all other parameters, suitable default values can be provided, e.g., $v_0 = 0$ for the initial multiplier guess, $B_0 = I$ for the initial estimate of the Hessian matrix of the Lagrangian function of (1) and $r_0 = (1, \ldots, 1)^T$ for the initial penalty parameters. In general it is assumed that $v_j^0 \geq 0$ for $j = m_e + 1, \ldots, m$, that B_0 is positive definite, and that all coefficients of the vector r_0 are positive.

The basic idea of the algorithm can be described in the following way: We determine a working set W_k and perform one step of a standard SQP-algorithm with respect to nonlinear programming problem with m_w nonlinear constraints. Then the working set is updated and the whole procedure repeated.

One particular advantage is that the numerical convergence conditions for the reduced problem are applicable for the original one as well, since all constraints not in the working set W_k are inactive, i.e., satisfy $g_j(x_k) > \epsilon$ for $j \in \{1, \ldots, m\} \setminus W_k$.

The line search procedure described in Schittkowski [10] can be used to determine a steplength parameter α_k, which is a combination of an Armijo-type steplength reduction with a quadratic interpolation of $\phi_k(\alpha)$. The proposed approach guarantees theoretical convergence results, is very easy to implement and works satisfactorily in practice. But in our case we want to achieve the additional requirement that all intermediate iterates $\alpha_{k,i}$ or $x_k + \alpha_{k,i-1} d_k$, respectively, do not possess more than m_w violated constraints. By introducing an additional loop reducing the steplength by a constant factor, it is always possible to guarantee this condition. An artificial penalty term is added to the objective function consisting of violated constraints. The modification of the line search procedure prevents iterates of the modified SQP-method that violate too many constraints.

BFGS-updates are standard technique in nonlinear programming and yield excellent convergence results both from the theoretical and numerical point of view. The modification to guarantee positive definite matrices B_k was proposed by Powell [9]. The update is performed with respect to the corrections $x_{k+1} - x_k$, and $\bigtriangledown_x L(x_{k+1}) - \bigtriangledown_x L(x_k)$.

Since a new restriction is included in the working set W_{k+1} only if it belongs to $J^\star_{k+1}$, we get always new and actual gradients in the quadratic programming subproblem (12). But gradients can be reevaluated for any larger set, e.g., W_{k+1}. In this case we can expect even a better performance of the algorithm.

The proposed modification of the standard SQP-technique is straightforward and easy to analyze. We want to stress out that its practical performance depends mainly on the heuristics used to determine the working set W_k. The first idea could be to take out those constraints from the working set which got the largest function values. However, the numerical size of a constraint depends on its internal scaling. In other words, we cannot conclude from a *large* restriction function value that the constraint is probably inactive.

To get a decision on constraints in the working set W_k that is independent of the scaling of the functions as much as possible, we propose the following rules:

- Among the constraints feasible at x_k and x_{k+1}, keep those in the working set that were violated during the line search. If there are too many of them according to some given constant, select constraints for which

$$\frac{g_j(x_{k+1}) - \epsilon}{g_j(x_{k+1}) - f_j(x_k + \alpha_{k,i-1} d_k)}$$

  is minimal. The decision whether a constraint is feasible or not, is performed with respect to the given tolerance ϵ.

- In addition keep the restriction in the working set for which $g_j(x_k + d_k)$ is minimal.

- Take out those feasible constraints from the working set, which are the *oldest* ones with respect to their successive number of iterations in the working set.

Under the assumptions mentioned so far, we can prove that

$$\phi'_{r_k}(0) = \bigtriangledown \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} < -\frac{1}{4}\gamma \|d_k\|^2 \tag{23}$$

for all k and a positive constant γ, see Schittkowski [13].

It is possible that we get a descent direction of the merit function, but that $\phi'_r(0)$ is extremely small. To avoid interruption of the whole iteration process, the idea is to

repeat the line search with another stopping criterion. Instead of testing (23), we accept a stepsize α_k as soon as the inequality

$$\phi_{r_k}(\alpha_k) \leq \max_{k-p(k)<=j<=k} \phi_{r_j}(0) + \alpha_k \mu \phi'_{r_k}(0) \tag{24}$$

is satisfied, where $p(k)$ is a predetermined parameter with $p(k) = \min\{k, p\}$, p a given tolerance. Thus, we allow an increase of the reference value $\phi_{r_k}(0)$ in a certain error situation, i.e., an increase of the merit function value. In case of $k = 0$, the reference value is adapted by a factor greater than 1, i.e., $\phi_{r_{j_k}}(0)$ is replace by $t\phi_{r_{j_k}}(0)$, $t > 1$. The basic idea to store reference function values and to replace the sufficient descent property by a sufficient 'ascent' property in max-form, see Dai and Schittkowski [2] for details and a convergence proof.

3 Numerical Tests

The modified SQP-algorithm is implemented in form of a Fortran subroutine with name NLPQLB. As pointed out in the previous section, an iteration consists of one step of a standard SQP-method, in our case of the code NLPQLP [17], with m_w constraints. Basically, only the definition of the working set and some rearrangements of index sets must be performed. Then NLPQLP is called to perform only one iteration proceeding from the iterates x_k, v_k, B_k, r_k and $J_k^\star$.

The algorithm as described in the previous section, requires an estimate of the maximum size of the working set, m_w, a starting point $x_0 \in I\!R^n$, and an initial working set W_0 with $J_0^\star \subset W_0$, see (13) for a definition of the active set $J_k^\star$. A straightforward idea is to sort the constraints according to their function values at x_0, and to take the first m_w constraints in increasing order. However, one would have to assume that all constraints are equally scaled, a very reasonable assumption in case of scalar semi-infinite problems of the the form (3).

Otherwise, an alternative, much simpler proposal could be to include all constraints in the initial working set for which $g_j(x) \leq \epsilon$, and to fill the remaining position with indices for which $g_j(x) > \epsilon$. A possible Fortran code fragment, where ACC corresponds to ϵ, is

```
IF (IFAIL.EQ.0) THEN
   I = 1
   K = MW
   DO J=1,M
      IF (G(J).LE.ACC) THEN
         KWA(I) = J
         I = I + 1
      ELSE
         IF ((K.GE.I).AND.(K.GT.0)) THEN
            KWA(K) = J
```

```
               K = K - 1
            ENDIF
         ENDIF
         IF (I-1.GT.MW) THEN
            WRITE(IOUT,*)' *** ERROR: Too many active ',
     /                                 'constraints at start!'
            STOP
         ENDIF
      ENDDO
   ENDIF
```

Here we assume that there are no equality constraints. Otherwise, they must be came part of the working set with $j \in W_0$ for all $1 \leq j \leq m_e$. Any other initialization of the working set depending on available information about the expected active constraints may be applied.

Some numerical experiments are reported to show that the resulting algorithm works as expected. The examples are small academic test problems taken from the literature, but somewhat modified in particular to get problems with a varying number of constraints. They have been used before to get the results published in Schittkowski [13], and are now solved with up to 200,000,000 instead of maximal 10,000 constraints. Gradients are evaluated analytically.

P1: The nonlinear semi-infinite test problem is taken from Tanaka, Fukushima, and Ibaraki [18],

$$x_1, x_2, x_3 \in I\!R: \quad \begin{array}{ll} \min & x_1^2 + x_2^2 + x_3^2 \\ & -x_1 - x_2 \exp(x_3 y) - \exp(2y) + 2\exp(4y) \geq 0 \quad \text{for all } y \in [0,1] \end{array} \tag{25}$$

with starting point $x_0 = (1, -1, 2)^T$. After a discretization of the interval $[0, 1]$ with $m = 4 \cdot 10^7$ equidistant points, we get a nonlinear program with 40,000,000 constraints. Figure 1 shows the curve plots over y for the starting point and the optimal solution $x^\star = (-0.21331259, -1.3614504, 1.8535473)^T$. Although only one constraint is active at the optimal solution $x^\star$, the starting point x_0 violates about 50 % of all constraints. Thus, the initial working set W_0 must be sufficiently large and we choose $m_w = 2 \cdot 10^7$. For the same reason, we restrict the discretization of the interval $[0, 1]$ and the total number of constraints by $m = 4 \cdot 10^7$.

P1F: This is the same nonlinear semi-infinite test problem as before. It is to be shown that the simple feasibility modification (11) by introducing an additional variable x_4 and a penalty term with $\rho = 10^4$ in the modified objective function reduces the number of

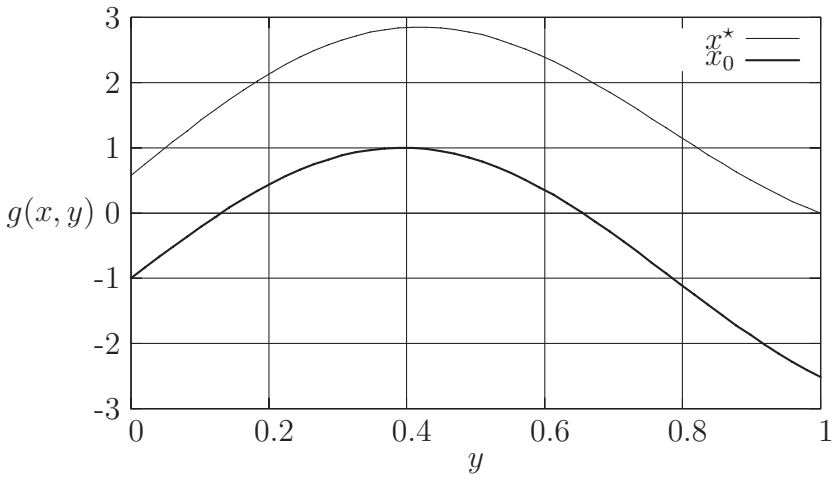

Figure 1: Function Plot for P1

intermediate active constraints significantly. The test problem is now given in the form

$$x_1, \ldots, x_4 \in I\!R : \begin{array}{ll} \min & x_1^2 + x_2^2 + x_3^2 + \rho x_4 \\ & -x_1 - x_2 \exp(x_3 y) - \exp(2y) + 2\exp(4y) \geq -x_4 \\ & \qquad \text{for all } y \in [0,1] \ , \\ & x_4 \geq 0 \ . \end{array} \tag{26}$$

The equidistant discretization is performed with $m = 2 \cdot 10^8$ points, and the initial iterate is $x_0 = (1, -1, 2, 100)^T$. No constraint is active at the starting point and we are able to choose a much smaller working set of size $m_w = 2 \cdot 10^3$.

P3: The nonlinear semi-infinite test problem is very similar to P1, see Tanaka, Fukushima, and Ibaraki [18],

$$x_1, x_2, x_3 \in I\!R : \begin{array}{ll} \min & \exp x_1 + \exp x_2 + \exp x_3 \\ & x_1 + x_2 y + x_3 y^2 - \dfrac{1}{1+y^2} \geq 0 \qquad \text{for all } y \in [0,1] \end{array} \tag{27}$$

with starting point $x_0 = (1, 0.5, 0)^T$. An equidistant discretization of the interval $[0,1]$ with $m = 2 \cdot 10^8$ equidistant points is chosen and the size of the working set is $m_w = 5 \cdot 10^5$. Figure 2 shows the curve plots over y for the starting point and the optimal solution $x^\star = (1.0066047, -0.12687988, -0.37972483)^T$. One constraint is active at the starting point and two at the optimal solution. However, we have to expect a larger number of nearly active constraints. Since the active set at the starting point differs significantly from the active set at the optimal solution, we have to choose a relatively large working set.

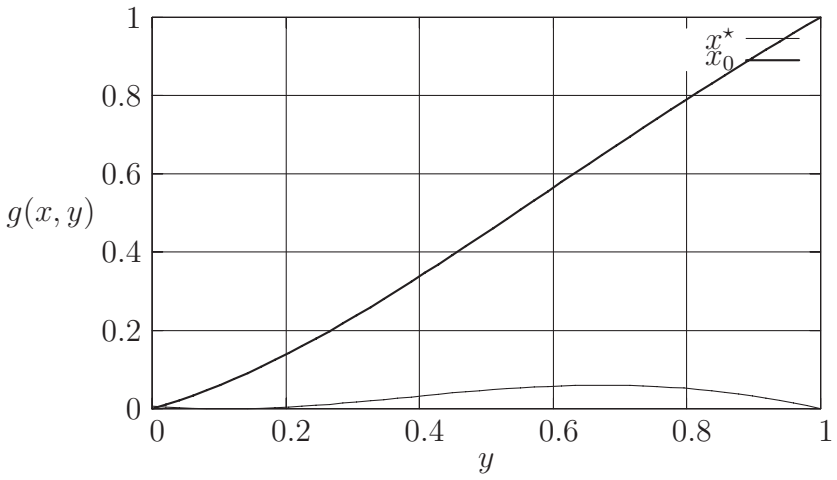

Figure 2: Function Plot for P3

P4: This is a nonlinear semi-infinite test problem with two free variables from the interval $[0,1] \times [0,1]$, see Tanaka, Fukushima, and Ibaraki [18],

$$\begin{aligned} & \min \quad x_1^2 + x_2^2 + x_3^2 \\ x_1, x_2, x_3 \in I\!R: \quad & -x_1(y_1 + y_2^2 + 1) - x_2 y_2 (y_1 - y_2) - x_3 y_2 (y_1 + y_2 + 1) \geq 1 \quad , \\ & \qquad\qquad \text{for all } y_1 \in [0,1] \text{ and } y_2 \in [0,1] \end{aligned} \tag{28}$$

with starting values $x_0 = (-2, -1, 0)^T$. Again, we use 200,000,000 uniformly distributed points of the interval $[0,1] \times [0,1]$ to discretize it. Since only one constraint is active for $y_1 = y_2 = 0$, the size of the working set can be as low as $m_w = 200$. Figure 3 shows the curvature plot over y_1 and y_2 for the optimal solution $x^\star = (-1, 0, 0)^T$.

TP332: The problem is a modification of the test problem TP332 of Schittkowski [12] to get a larger number of constraints,

$$\begin{aligned} & \min \quad \sum_{i=1}^{m} \Big((\log t_i + x_2 \sin t_i + x_1 \cos t_i)^2 + (\log t_i + x_2 \cos t_i - x_1 \sin t_i)^2 \Big) \\ x_1, x_2 \in I\!R: \quad & \arctan\left(\frac{1/t_i - x_1}{\log t_i + x_2} \right) \leq \frac{\pi}{60} \quad , \quad i = 1, \ldots, m \end{aligned} \tag{29}$$

with starting values $x_0 = (0.75, 0.75)^T$, where $t_i \doteq \pi(1/3 + (i-1)/180)$. We proceed from $m = 2 \cdot 10^8$ constraints, and the size of the working set is $m_w = 100$. Only one constraint is active at the starting and the optimal solution $x^\star = (0.94990963, 0.049757167)^T$.

TP374: The problem is extended in a straightforward way to allow more than 35 con-

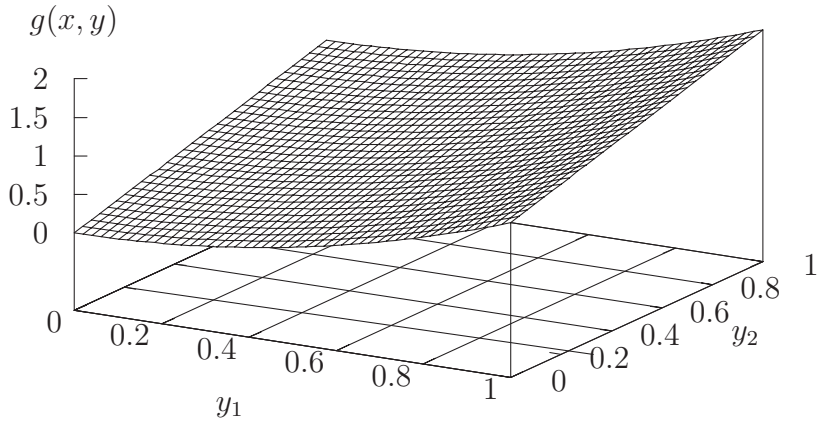

Figure 3: Function Plot for P4

straints given in Schittkowski [12],

$$x \in I\!R^{10}: \quad \begin{array}{rcll} \min & x_{10} & & \\ z(t_i) - (1-x_{10})^2 & \geq & 0 \ , & i = 1, \ldots, r \ , \\ -z(t_i) + (1+x_{10})^2 & \geq & 0 \ , & i = r+1, \ldots, 2r \ , \\ -z(t_i) + x_{10}^2 & \geq & 0 \ , & i = 2r+1, \ldots, 3.5r \ , \end{array} \tag{30}$$

where

$$z(t) \doteq \left(\sum_{k=1}^{9} x_k \cos(kt) \right)^2 + \left(\sum_{k=1}^{9} x_k \sin(kt) \right)^2$$

and

$$\begin{array}{rcll} t_i & = & \pi(i-1)0.025 \ , & i = 1, \ldots, r \ , \\ t_i & = & \pi(i-1-r)0.025 \ , & i = r+1, \ldots, 2r \ , \\ t_i & = & \pi(1.2 + (i-1-2r)0.2)0.25 \ , & i = 2r+1, \ldots, 3.5r \ . \end{array}$$

Starting solution is $x_0 = (0.1, 0.1, \ldots, 0.1, 1)^T$. By choosing $r = 10^8/3.5$, we get 100,000,000 nonlinear constraints. Because of a large number of intermediate active constraints, we have to restrict the total number of constraints and let $m_w = 2 \cdot 10^6$. The optimal solution is

$$\begin{array}{rcl} x^\star & = & (0.25559274, \quad 0.25291560, \quad 0.29120000, \quad 0.26826036, \quad 0.18929502, \\ & & 0.082428568, \quad -0.014405861, \quad -0.070673846, \quad -0.18878711, \quad 0.29170025)^T \ . \end{array}$$

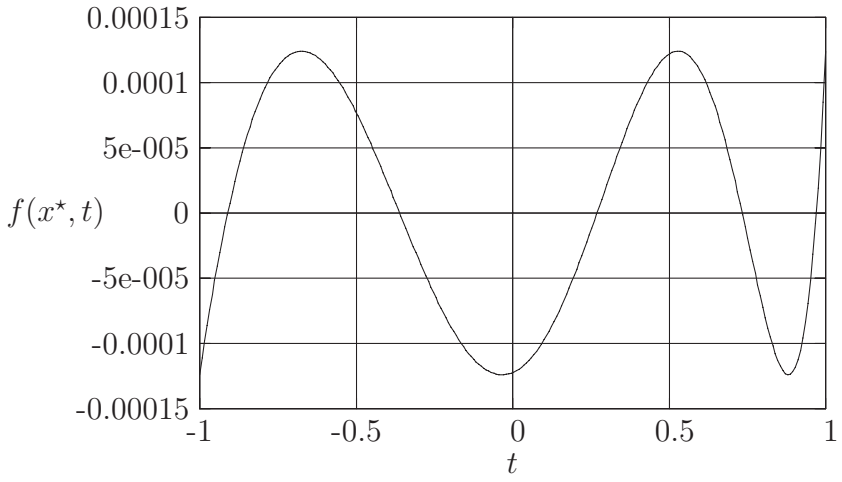

Figure 4: Function Plot for U3

U3: The goal is to approximate the exponential function by a rational one, i.e., to minimize the maximum norm of r functions, see Luksan [7],

$$\min_{x \in I\!R^5} \max\{|\frac{x_1 + x_2 t_i}{1 + x_3 t_i + x_4 t_i^2 + x_5 t_i^3} - \exp(t_i)| \ , \quad i = 1, \ldots, r\} \ , \tag{31}$$

where

$$t_i \doteq 2\frac{i-1}{r-1} - 1$$

for $i = 1, \ldots, r$. Starting point is $x_0 = (0.5, 0, 0, 0, 0)^T$. The problem is transformed into a smooth nonlinear program of the form (8) with $m \doteq 2r$ constraints and $n + 1$ variables. The starting point for the additional variable is set to $t = 20$, and the number of constraints is $r = 5 \cdot 10^7$ or $m = 10^8$, respectively, where we expect a large number of intermediate active constraints. Thus, we allow up to $m_w = 5 \cdot 10^5$ constraints in the working set. Figure 4 shows the curvature plot of the residual function $f(x, t)$ as defined by (31) over t for the optimal solution

$$x^\star = (0.99987768, 0.25365090, -0.74654084, 0.24513247, -0.037465273)^T \ .$$

L5: The problem is similar to the previous one. Again, the sum of absolute values of a set of r differentiable functions is to be minimized, see Luksan [7], but now with additional

linear equality and inequality constraints,

$$
\begin{array}{rlcl}
 & \min \quad \max & & \{|1+2\sum_{j=1}^{7}\cos(2\pi x_j \sin\theta_i)|\ , \quad i=1,\ldots,r\} \\
 & -x_4+x_6 & = & 1\ , \\
 & x_7 & = & 3.5\ , \\
x\in I\!R^7: & -x_1+x_2 & \geq & 0.4\ , \\
 & -x_2+x_3 & \geq & 0.4\ , \\
 & -x_3+x_4 & \geq & 0.4\ , \\
 & -x_4+x_5 & \geq & 0.4\ , \\
 & -x_5+x_6 & \geq & 0.4\ , \\
 & -x_6+x_7 & \geq & 0.4\ ,
\end{array}
\tag{32}
$$

where

$$\theta_j \doteq \frac{\pi}{180}(8.5+0.5i)$$

for $i = 1, \ldots, r$. Starting point is $x_0 = (0.5, 1, 1.5, 2, 2.5, 3, 3.5)^T$. The problem is transformed into a smooth nonlinear program (8) with $m \doteq 2r + 8$ constraints and $n + 1$ variables. The starting point for the additional variable is set to $t = 1$. We set $r = 10^8 - 4$ leading to $m = 2 \cdot 10^8$ constraints. The number of constraints in the working set is $m_w = 4 \cdot 10^4$. Figure 5 shows the curvature plot of the residual function $f(x, t)$ as defined by (32) over θ for the optimal solution

$$x^\star = (0.34626538, 0.75414273, 1.2127664, 1.7308550, 2.1702278, 2.7308550, 3.5)^T\ .$$

We observe a larger number of non-connected active constraints at the optimal solution.

E5: Again, we minimize the maximum norm of r functions, see Hald and Madsen [4],

$$\min_{x\in I\!R^4} \max\{(x_1 + x_2 t_i - \exp t_i)^2 + (x_3 + x_4 \sin t_i - \cos t_i)^2\ , \quad i = 1, \ldots, r\} \tag{33}$$

where

$$t_i \doteq \frac{4i}{r}$$

for $i = 1, \ldots, r$. Starting point is $x_0 = (25, 5, -5, -1)^T$. The problem is transformed into a smooth nonlinear program of the form (8) with $m \doteq r$ constraints and $n + 1$ variables. The starting point for the additional variable is set to $t = 1,000$. For $r = 2 \cdot 10^8$ we get $m = 2 \cdot 10^8$ constraints, and the number of constraints in the working set is $m_w = 5 \cdot 10^4$. Figure 6 shows the curvature plot of the residual function $f(x, t)$ as defined by (33) over t for the optimal solution

$$x^\star = (-10.112611, 13.376871, -0.45892449, -0.17124647)^T\ .$$

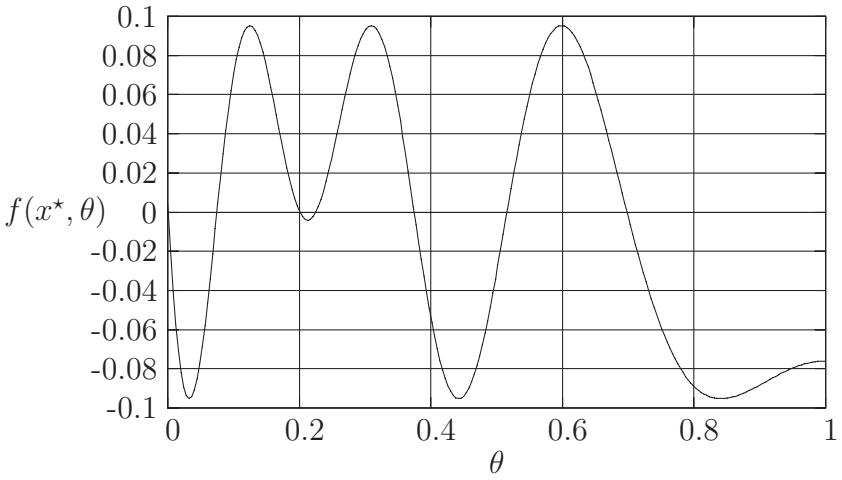

Figure 5: Function Plot for L5

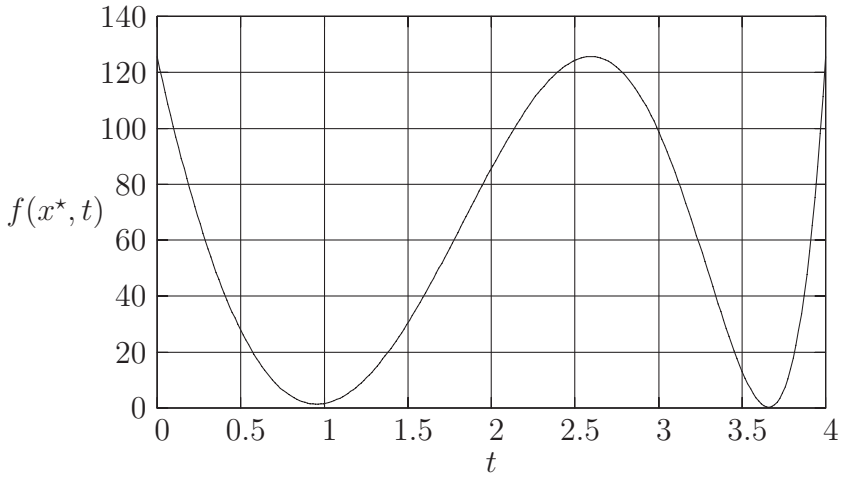

Figure 6: Function Plot for E5

name	n	m	m_w	$f^\star$
P1	3	40,000,000	20,000,000	5.33469
P1F	4	200,000,000	2,000	5.33469
P3	3	200,000,000	500,000	4.30118
P4	3	200,000,000	200	1.00000
TP332	2	200,000,000	100	398.587
TP374	10	100,000,000	2,000,000	0.434946
U3	6	100,000,000	500,000	0.00012399
L5	8	200,000,000	40,000	0.0952475
E5	5	200,000,000	50,000	125.619

Table 1: Test Examples

The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 10.1, EM64T, under Windows Vista and Intel(R) Core(TM) Duo CPU E8500, 3.16 GHz, and 8 GB RAM. The working arrays of the routine calling NLPQLB are dynamically allocated. Quadratic programming subproblems are solved by the primal-dual method of Goldfarb and Idnani [3] based on numerically stable orthogonal decompositions. NLPQLB is executed with termination accuracy $\epsilon = 10^{-8}$. Numerical experiments are reported in Table 2 where we use the following notation:

name	-	identification of test example
n	-	number of variables in standard form (1)
m	-	total number of constraints in standard form (1)
m_w	-	number of constraints in the working set
$f^\star$	-	final objective function value
$\|J_{max}\|$	-	maximum number of active constraints
n_f	-	number of simultaneous function computations, i.e., of objective function and all constraints at a given iterate
n_g^{tot}	-	total number of gradient computations, i.e., of all individual constraint gradient evaluations
n_{its}	-	number of iterations or simultaneous gradient computations, i.e., of gradients of objective function and all constraints at a given iterate
time	-	calculation time in seconds
$f^\star$	-	final objective function value

It is obvious that the efficiency of an active set strategy strongly depends on how close the active set at the starting point to that of the optimal solution is. If dramatic changes of active constraints are expected as in case of P1, i.e., if intermediate iterates with a large number of violated constraints are generated, the success of the algorithm is marginal.

name	$\lvert J_{max}\rvert$	n^f	n_g^{tot}	n_{its}	t_{calc}
P1	5,579,011	20	27,597,939	13	97
P1F	801	23	6,728	15	283
P3	129,237	12	1,558,343	10	77
P4	1	4	203	4	16
TP332	1	12	110	11	464
TP374	584,004	150	20,238,000	89	3,361
U3	285,901	191	5,145,498	58	903
L5	39,976	80	207,960	24	1,224
E5	41,674	30	212,245	21	392

Table 2: Numerical Results

On the other hand, practical optimization problems often have special structures from where good starting points can be predetermined. Examples P1F and especially P4 and TP332 show a dramatic reduction of derivative calculations, which is negligible compared to the number of function calls.

Since the constraints are nonlinear and non-convex, we have to compute all m constraint function values at each iteration to check feasibility and to predict the new active set. The total number of individual constraint function evaluations is $n_f \cdot m$.

Calculation times are excessive and depend mainly on data transfer operations from and to the standard swap file of Windows, and the available memory in core, which is 4 GB in our case. To give an example, test problem TP332 requires 42 sec for $m = 2 \cdot 10^7$ constraints and less than 3 sec for $m = 2 \cdot 10^6$ constraints.

Note that the code NLPQLB requires additional working space in the order of $2m$ double precision real numbers plus $m_w \cdot (n + 1)$ double precision numbers for the partial derivatives of constraints in the working set. Thus, the total memory to run a test problem with $m = 2 \cdot 8$ constraints requires at least 600,000,000 double precision numbers and in addition at least $400,000,000$ logical values.

It is amazing that numerical instabilities due to degeneracy are prevented. The huge number of constraints indicates that the derivatives are extremely close to each other, making the optimization problem unstable. The constraint qualification, i.e., the linear independence of active constraints, is more or less violated. We benefit from the fact that derivatives are analytically given.

4 Program Documentation

NLPQLB is implemented in form of a Fortran subroutine, which computes appropriate indices of constraints belonging to the working set. A nonlinear program with the same objective function, but a reduced set of m_w constraints is formulated and solved by one

iteration of NLPQLP, see Schittkowski [17]. Model functions and gradients are called by reverse communication. The user has to provide functions and gradients in the same program which executes NLPQLB, according to the following rules:

1. Choose starting values for the variables to be optimized, and store them in X.
2. Compute objective and all constraint function values, store them in F and G, respectively.
3. Predetermine the initial working set which must contain all equality constraints and at least all violated inequality constraints, and store the indices in the first MW positions of the integer array KWA.
4. Compute gradients of objective function and all constraints, and store them in DF and DG, respectively. The J-th row of DG contains the gradient of the KWA(J)-th constraint, J=1,...,MW.
5. Set IFAIL=0 and execute NLPQLB.
6. If NLPQLB returns with IFAIL=-1, compute objective function and constraint values for all variable values in X, store them in F and G, and call NLPQLB again.
7. If NLPQLB terminates with IFAIL=-2, compute gradient values with respect to the variable values in X, and store them in DF and DG. Only derivatives for active constraints, ACTIVE(KWA(J))=.TRUE., need to be computed and stored in the J-th row of DG, J=1,...,MW. Then call NLPQLB again.
8. If NLPQLB terminates with IFAIL=0, the internal stopping criteria are satisfied. In case of IFAIL>0, an error occurred.

Here, MW (=m_w) is a user provided guess for the maximum number of expected violated constraints with $n \leq m_w \leq m$.

Usage:

```
    CALL  NLPQLB (      M,      ME,      MW,   MWMAX,       N,
   /                 NMAX,    MNN2,       X,       F,       G,
   /                   DF,      DG,       U,      XL,      XU,
   /                    C,       D,     ACC,   ACCQP,  MAXFUN,
   /                MAXIT,   MAXNM,    RHOB,  IPRINT,    IOUT,
   /                 IFAIL,      WA,     LWA,     KWA,    LKWA,
   /                  ACT,    LACT,  QPSLVE                    )
```

Definition of the parameters:

M :	Total number of constraints.
ME :	Number of equality constraints.
MW :	Size of working set. MW must be at least one and not greater than M. If M is greater than N, MW must be greater than N. Otherwise, MW must be equal to M and is a guess for the maximum number of expected active constraints.
MWMAX :	Row dimension of DG. MW must not be greater than MWMAX. If M is greater than N, MWMAX must be at least N.
N :	Number of optimization variables.
NMAX :	Row dimension of C. NMAX must be at least two and greater than N.
MNN2 :	Must be equal to MW+N+N+2 when calling NLPQLP.
X(NMAX) :	Initially, X has to contain starting values for the optimal solution. On return, X is replaced by the current iterate. In the driving program the dimension of X has to be equal to NMAX.
F :	On return, F contains the final objective function value.
G(M) :	On return, G contains the constraint function values at the final iterate X. In the driving program, the dimension of G has to at least M.
DF(NMAX) :	DF contains the current gradient of the objective function.
DG(MWMAX,NMAX) :	DG contains the gradients of the active constraints (ACTIVE(K(J))=.TRUE., J=1,...,MW) at a current iterate X. The remaining rows are filled with previously computed gradients. In the driving program, the row dimension of DG has to be equal to MWMAX.
U(MNN2) :	U contains the multipliers with respect to the actual iterate stored in X. The first MW locations contain the multipliers of the MW nonlinear constraints belonging to the working set, the subsequent N locations the multipliers of the lower bounds, and the final N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.

XL(N),XU(N) :	On input, the one-dimensional arrays XL and XU must contain the lower and upper bounds of the variables, respectively.
C(NMAX,NMAX) :	On return, C contains the last computed approximation of the Hessian matrix of the Lagrangian function. In the driving program, the row dimension of C has to be equal to NMAX.
D(NMAX) :	Auxiliary array for linear part in the QP.
ACC :	The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be much smaller than the accuracy by which gradients are computed.
ACCQP :	The tolerance is needed for the QP solver to perform several tests, for example whether optimality conditions are satisfied or whether a number is considered as zero or not. If ACCQP is less or equal to zero, then the machine precision is computed by NLPQLB and subsequently multiplied by 1.0D+4.
MAXFUN :	The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20). MAXFUN must not be greater than 50.
MAXIT :	Maximum number of outer iterations, where one iteration corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g. 100).
MAXNM :	Stack size for storing merit function values at previous iterations for non-monotone line search (e.g. 10). If MAXNM=0, monotone line search is performed. MAXNM should not be greater than 50.
RHOB :	Parameter for initializing a restart in case of IFAIL=2 by setting the BFGS-update matrix to rhob*I, where I denotes the identity matrix. The number of restarts is bounded by MAXFUN. No restart is performed if RHOB is set to zero. Must be non-negative (e.g. 100).

IPRINT : Specification of the desired output level.

0 - No output of the program.

1 - Only final convergence analysis.

2 - One line of intermediate results for each iteration.

3 - More detailed information for each iteration.

4 - More line search data displayed.

IOUT : Integer indicating the desired output unit number, i.e., all write-statements start with 'WRITE(IOUT,... '.

IFAIL : The parameter shows the reason for terminating a solution process. Initially, IFAIL must be set to zero. On return, IFAIL could contain the following values:

-2 - Compute new gradient values.

-1 - Compute new function values.

0 - Optimality conditions satisfied.

1 - Stop after MAXIT iterations.

2 - Uphill search direction.

3 - Underflow when computing new BFGS-update matrix.

4 - Line search exceeded MAXFUN iterations.

5 - Length of a working array too short.

6 - False dimensions, e.g., M<MW, N≥NMAX, MNN2≠MW+N+N+2, etc.

7 - Search direction close to zero at infeasible iterate.

8 - Starting point violates lower or upper bound.

9 - Wrong input parameter, e.g., MODE, IPRINT, IOUT.

10 - Inconsistency in QP, division by zero.

11 - There are too many active constraints, increase MW.

>100 - Error message of QP solver.

WA(LWA) : WA is a real working array of length LWA.

LWA : Length of the real working array WA. LWA must be at least at least 23*N+2*M+7*MWMAX+150.

NOTE: The standard QP-solver coming together with NLPQLP (QL) needs additional memory for 3*NMAX*NMAX/2+10*NMAX+2*MWMAX+1 real numbers.

KWA(LKWA) : KWA is an integer working array of length LKWA.

LKWA : Length of the integer working array KWA. LKWA should be at least 2*MW+max(N+1,MW/NMAX)+25. The first MW positions in KWA are used for storing the working set. When starting NLPQLB, KWA has to contain a set of MW indices of constraints for the first working set including all constraints active at the starting point X, i.e., for which G(J) is less than ACC, j=1,...,M. Remaining positions of G are filled with indices belonging to inactive constraints, i.e., for which G(J) is greater or equal to ACC for J=1,...,M.

NOTE: The standard QP-solver coming together with NLPQLP (QL) needs additional memory for N integer numbers.

ACTIVE(LACTIV) : The logical array indicates constraints, which NLPQLB considers to be active at the last computed iterate, i.e., G(J) is active, if and only if ACTIVE(K(J)) is true for J=1,...,MW.

LACTIV : Length of the logical array ACTIVE. The length of the logical array should be at least 2*M + 2*MW + 11.

QPSLVE : External subroutine to solve the quadratic programming subproblem. The calling sequence is

```
CALL QPSLVE(M,ME,MWMAX,N,NMAX,MNN,C,D,A,B,
/          XL,XU,X,U,EPS,MODE,IOUT,IFAIL,IPRINT,
/          WAR,LWAR,IWAR,LIWAR)
```

For more details about the choice and dimensions of arguments, see [15].

Some of the termination reasons depend on the accuracy used for approximating gradients. If we assume that all functions and gradients are computed within machine precision and that the implementation is correct, there remain only the following possibilities that could cause an error message:

1. The termination parameter ACC is too small, so that the numerical algorithm plays around with round-off errors without being able to improve the solution. Especially the Hessian approximation of the Lagrangian function becomes unstable in this case. A straightforward remedy is to restart the optimization cycle again with a larger stopping tolerance.

2. The constraints are contradicting, i.e., the set of feasible solutions is empty. There is no way to find out whether a general nonlinear and non-convex set possesses a feasible point or not. Thus, the nonlinear programming algorithms will proceed

until running in any of the mentioned error situations. In this case, the correctness of the model must be very carefully checked.

3. Constraints are feasible, but some of them there are degenerate, for example if some of the constraints are redundant. One should know that SQP algorithms assume the satisfaction of the so-called constraint qualification, i.e., that gradients of active constraints are linearly independent at each iterate and in a neighborhood of an optimal solution. In this situation, it is recommended to check the formulation of the model constraints.

However, some of the error situations also occur if, because of wrong or inaccurate gradients, the quadratic programming subproblem does not yield a descent direction for the underlying merit function. In this case, one should try to improve the accuracy of function evaluations, scale the model functions in a proper way, or start the algorithm from other initial values.

NLPQLB returns the best iterate obtained. In case of successful termination (IFAIL=0), this is always the last iterate, in case of non-successful return (IFAIL>0) an eventually better previous iterate. The success if measured by objective function value and constraint violation. Note that the output of constraints and multiplier values is suppressed for $m > 10,000$.

The QP solver is defined in form of an external subroutine to allow a replacement in case of exploiting special sparsity patterns. A typical example is the usage of NLPQLP for solving least squares problems, where artificially introduced equality constraints lead to a Jacobian which consist partially of the identity matrix, see Schittkowski [14, 16].

5 Example

To give an example how to organize the code, we consider test example (27), i.e., problem P3,

$$\begin{aligned} & \min \quad \exp(x_1) + \exp(x_2) + \exp(x_3) \\ x_1, x_2 \in I\!R: \quad & x_1 + x_2 y + x_3 y^2 - \frac{1}{1+y^2} \geq 0 \\ & \text{for all } y \in [0,1] \quad . \end{aligned} \tag{34}$$

The Fortran source code for executing NLPQLP is listed below. Gradients are given in analytical form. The function block inserted in the main program can be replaced by a subroutine call. Also the gradient evaluation is easily exchanged by a difference formula.

```
IMPLICIT  NONE
INTEGER   NMAX,MMAX,MWMAX,MNN2X,LWA,LKWA,LACTIV
PARAMETER (NMAX = 4, MMAX = 10000, MWMAX = 20)
```

```
      PARAMETER (MNN2X = MWMAX + NMAX + NMAX + 2,
     /           LWA = 1.5*NMAX*NMAX + 9*MWMAX + 33*NMAX + 2*MMAX + 150,
     /           LKWA = NMAX + 2*MWMAX + MAX(NMAX+1,MWMAX/NMAX) + 25,
     /           LACTIV = 2*MMAX + 2*MWMAX + 11)
      INTEGER   KWA(LKWA),N,ME,M,MW,MNN2,MAXIT,MAXFUN,IPRINT,
     /          MAXNM,IOUT,IFAIL,I,J,K
      DOUBLE PRECISION X(NMAX),F,G(MMAX),DF(NMAX),
     /          DG(MWMAX,NMAX),U(MNN2X),XL(NMAX),XU(NMAX),C(NMAX,NMAX),
     /          D(NMAX),WA(LWA),ACC,ACCQP,RHOB,Y
      LOGICAL   ACTIVE(LACTIV)
      EXTERNAL  QL
C
C   Set some constants and initial values
C
      ACC    = 1.0D-10
      ACCQP  = 1.0D-12
      MAXIT  = 500
      MAXFUN = 20
      MAXNM  = 20
      RHOB   = 0.1D0
      IPRINT = 2
      N      = NMAX - 1
      MW     = MWMAX
      M      = MMAX
      ME     = 0
      MNN2   = MW + N + N + 2
      IFAIL  = 0
      IOUT   = 6
C
C ... initialize problem data
C
      DO I=1,N
         XL(I) = -100.0D0
         XU(I) = 100.0D0
      ENDDO
      X(1) = 1.0D0
      X(2) = 0.5D0
      X(3) = 0.0D0
C
C=============================================================
C
C ... compute function values
C
    1 CONTINUE
```

```
      F = DEXP(X(1)) + DEXP(X(2)) + DEXP(X(3))
      DO J=1,M
         Y = DBLE(J-1)/DBLE(M-1)
         G(J) =  X(1) + X(2)*Y + X(3)*Y**2 - 1.0D0/(1.0D0+Y**2)
      ENDDO
      IF (IFAIL.EQ.-1) GOTO 4
C
C=============================================================
C
C ... determine initial working set
C
      I = 1
      K = MW
      DO J=1,M
         IF (IFAIL.EQ.0) THEN
            ACTIVE(J) = .TRUE.
            IF (G(J).LE.ACC) THEN
               KWA(I) = J
               I = I + 1
            ELSE
               IF ((K.GE.I).AND.(K.GT.0)) THEN
                  KWA(K) = J
                  K = K - 1
               ENDIF
            ENDIF
            IF (I-1.GT.MW) THEN
               WRITE(IOUT,*)' *** ERROR: Too many active ',
     /                                    'constraints at start!'
               STOP
            ENDIF
         ENDIF
      ENDDO
C
C=============================================================
C
C   ... compute partial derivatives
C
    2 CONTINUE
      DO I=1,N
         DF(I) = DEXP(X(I))
      ENDDO
      DO I=1,MW
         J = KWA(I)
         IF (ACTIVE(J)) THEN
```

```
            Y = DBLE(J-1)/DBLE(M-1)
            DG(I,1) = 1.0D0
            DG(I,2) = Y
            DG(I,3) = Y**2
          ENDIF
        ENDDO
        IF (IFAIL.EQ.-2) GOTO 4
C
C=============================================================
C
C   ... call NLPQLB
C
    4 CONTINUE
      CALL NLPQLB(M,ME,MW,MWMAX,N,NMAX,MNN2,X,F,G,DF,DG,U,XL,XU,
     /     C,D,ACC,ACCQP,MAXFUN,MAXIT,MAXNM,RHOB,IPRINT,
     /     IOUT,IFAIL,WA,LWA,KWA,LKWA,ACTIVE,LACTIV,QL)
C
C   ... repeat, if necessary
C
      IF (IFAIL.EQ.-1) GOTO 1
      IF (IFAIL.EQ.-2) GOTO 2
C
      STOP
      END
```

The following output is generated:

```
 ---------------------------------------------------------------------
  START OF THE SEQUENTIAL QUADRATIC PROGRAMMING ALGORITHM
 ---------------------------------------------------------------------

  Parameters:
     N      =       3
     M      =      20
     ME     =       0
     MODE   =       0
     ACC    =   0.1000D-09
     ACCQP  =   0.1000D-11
     STPMIN =   0.1000D-09
     MAXFUN =      20
     MAXNM  =      20
     MAXIT  =     500
     IPRINT =       2
```

```
 Output in the following order:
    IT    - iteration number
    F     - objective function value
    SCV   - sum of constraint violations
    NA    - number of active constraints
    I     - number of line search iterations
    ALPHA - steplength parameter
    DELTA - additional variable to prevent inconsistency
    KKT   - Karush-Kuhn-Tucker optimality criterion

  IT         F           SCV      NA  I    ALPHA     DELTA     KKT
-----------------------------------------------------------------------
   1  0.53670031D+01  0.00D+00   20  0  0.00D+00  0.00D+00  0.37D+01
   2  0.50225530D+01  0.00D+00    7  2  0.10D+00  0.00D+00  0.14D+01
   3  0.42955784D+01  0.28D-02    8  1  0.10D+01  0.00D+00  0.11D-01
   4  0.43011457D+01  0.34D-04    8  1  0.10D+01  0.00D+00  0.58D-04
   5  0.43011576D+01  0.43D-04   13  2  0.50D+00  0.00D+00  0.49D-04
   6  0.43011820D+01  0.34D-05    9  1  0.10D+01  0.00D+00  0.31D-05
   7  0.43011835D+01  0.53D-06    8  1  0.10D+01  0.00D+00  0.56D-06
   8  0.43011838D+01  0.28D-15    3  1  0.10D+01  0.00D+00  0.53D-15

  --- Final Convergence Analysis at Best Iterate ---

    Best result at iteration:      ITER  =       8
    Objective function value:      F(X)  =  0.43011838D+01
    Approximation of solution:     X     =
       0.10066048D+01 -0.12688079D+00 -0.37972400D+00
    Approximation of multipliers:  U     =
       0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00  0.66070270D+00  0.00000000D+00
       0.00000000D+00  0.10118733D+01  0.10637190D+01  0.00000000D+00
       0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00
    Constraint values:             G(X)  =
       0.82441438D-05  0.49961983D-06  0.23264920D-05  0.86786617D-05
       0.91243683D-05  0.95812657D-05  0.10049356D-04  0.33291179D-07
       0.66587168D-07  0.16635954D-06  0.11098662D-06  0.16649195D-06
       0.23310554D-06  0.31082979D-06 -0.55511151D-16  0.39966709D-06
       0.66179110D-05 -0.11102230D-15 -0.11102230D-15  0.11096258D-07
    Distance from lower bound:     XL-X  =
      -0.10100660D+03 -0.99873119D+02 -0.99620276D+02
```

```
Distance from upper bound:     XU-X  =
   0.98993395D+02  0.10012688D+03  0.10037972D+03
Number of function calls:      NFUNC =      10
Number of gradient calls:      NGRAD =       8
Number of calls of QP solver:  NQL   =       8


Active constraints:  KWA =
       1023      1052      1082      1022
       1021      1020      1019      1059
       1058      1067      1057      1056
       1055      1054     10000      1053
       1027      1062      1061      1060
```

6 Conclusions

We present a modification of an SQP algorithm to solve optimization problems with a very large number of constraints, m, relative to the number of variables, n. The idea is to proceed from a user-provided guess, m_w, for the maximum number of violated constraints, and to solve quadratic programming subproblems with m_w linear constraints instead of m constraints.

Some numerical experiments with simple academic test problems show that it is possible to solve problems with up to $m = 2 \cdot 10^8$ nonlinear constraints, which would not be solvable otherwise by a standard SQP algorithm. Sparsity of the Jacobian of the constraints is not assumed.

The performance depends significantly on the position of the starting point, i.e., on the question, how close the initial active set is to the active set at the final solution. If, in the worst case, all constraints are violated at an intermediate iterate, the proposed active set strategy is useless, as probably any other as well. In practical applications, however, optimization problems are often routinely solved and some information about the choice of good starting points is available. If there are no or very little changes of the active set and if only a few constraints are active, the achievements are significant.

Is is very amazing that it possible at all to solve problems with this huge number of constraints on a standard PC. The test examples have a quite simple structure, in most cases arising from a semi-infinite optimization and an equidistant discretization. It must be expected that gradients of neighbored constraints coincide up to seven digits. These optimization problems are highly unstable in the sense that the linear independency constraint qualification is more or less violated at all iterates.

References

[1] Byrd R., Gould N., Nocedal J., Waltz R. (2004): *An Active-Set Algorithm for Nonlinear Programming Using Linear Programming and Equality Constrained Subproblems*, Mathematical Programming B, Vol. 100, 27 - 48, with R. Byrd, N. Gould and R. Waltz, 2004

[2] Dai Y.H., Schittkowski K. (2006): *A sequential quadratic programming algorithm with non-monotone line search*, Pacific Journal of Optimization, Vol. 4, 335-351

[3] Goldfarb D., Idnani A. (1983): *A numerically stable method for solving strictly convex quadratic programs*, Mathematical Programming, Vol. 27, 1-33

[4] Hald J., Madsen K. (1981): *Combined LP and quasi-Newton methods for minmax optimization*, Mathematical Programming, Vol. 20, 49-62

[5] Kneppe G. (1990): *MBB-LAGRANGE: Structural optimization system for space and aircraft structures*, Report S-PUB-0406, MBB Hubschrauber und Flugzeuge, Ottobrunn, Munich

[6] Lenard M. (1979): *A computational study of active set strategies in nonlinear programming with linear constraints* , Mathematical Programming, Vol. 16, 81 - 97

[7] Luksan L. (1985): *An implementation of recursive quadratic programming variable metric methods for linearly constrained nonlinear minmax approximations*, Kybernetika, Vol 21, 22-40

[8] Ortega J.M., Rheinbold W.C. (1970): *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York-San Francisco-London

[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

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

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

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

[13] Schittkowski K. (1992): *Solving nonlinear programming problems with very many constraints*, Optimization, Vol. 25, 179-196

[14] Schittkowski K. (2002): *Numerical Data Fitting in Dynamical Systems*, Kluwer Academic Publishers, Dordrecht

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

[16] Schittkowski K. (2003): *DFNLP: A Fortran implementation of an SQP-Gauss-Newton algorithm - user's guide*, Report, Department of Mathematics, University of Bayreuth, 2003

[17] Schittkowski K. (2006): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user's guide, version 2.2*, Report, Department of Computer Science, University of Bayreuth

[18] Tanaka Y., Fukushima M., Ibaraki T. (1988):*A comparative study of several semi-infinite nonlinear programming algorithms*, European Journal of Operations Research, Vol. 36, 92-100

[19] Wolfe P. (1969): *Convergence conditions for ascent methods,* SIAM Review, Vol. 11, 226-235