IMPLEMENTATION OF A SEQUENTIAL QUADRATIC PROGRAMMING CODE FOR PARALLEL COMPUTING

K. SCHITTKOWSKI

Department of Mathematics, University of Bayreuth, 95440 Bayreuth, Germany

...

The paper introduces a new version of the SQP code NLPQL, which is widely used in commercial and academic institutions to solve smooth nonlinear programming problems. The new version, NLPQLP, is specifically tuned to run under distributed systems. Another input parameter l is introduced for the number of parallel machines, that is the number of function calls to be executed simultaneously. In case of $l = 1$, NLPQLP is identical to NLPQL. Otherwise, the line search procedure is modified to allow parallel function calls, which can also be applied for approximating gradients by difference formulae. The mathematical background is outlined, in particular the modification of the line search algorithm to retain convergence under parallel systems. Numerical results show the sensitivity of the new version with respect to the number of parallel machines and the influence of different gradient approximations under uncertainty. The performance evaluation is obtained by more than 300 standard test problems. It must be emphasized that the distributed computation of function values is only simulated throughout the paper. It is up to the user to adopt the code to a particular parallel environment.

KEY WORDS: nonlinear optimization, nonlinear programming, sequential quadratic programming, SQP, line search, parallel computing, distributed computing, performance evaluation, numerical tests

1 INTRODUCTION

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

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

where x is an n-dimensional parameter vector. To facilitate the subsequent notation, we assume that upper and lower bounds x_u and x_l are not handled separately, i.e., we consider the somewhat simpler formulation

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

It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on the whole $\mathbb{R}^n$. But besides of this we do not suppose any further mathematical structure of the model functions.

Sequential quadratic programming is the standard general purpose method to solve smooth nonlinear optimization problems, at least under the following assumptions:

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

The code NLPQL of Schittkowski[29] is a Fortran implementation of a sequential quadratic programming (SQP) algorithm. The design of the numerical algorithm is founded on extensive comparative numerical tests of Schittkowski[22,26,24], Schittkowski et al.[37], Hock and Schittkowski[14], and on further theoretical investigations published in [23,25,27,28]. The algorithm is extended to solve also nonlinear least squares problems efficiently, see [31] or [35], and to handle problems with very many constraints, cf. [32]. To conduct the numerical tests, a random test problem generator is developed for a major comparative study, see [22]. Two collections with more than 300 academic and real-life test problems are published in Hock and Schittkowski[14] and in Schittkowski[30]. Fortran source codes and a test frame can be downloaded from the home page of the author,

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

The test examples are part of the Cute test problem collection of Bongartz et al.[4]. More than 100 test problems based on a Finite Element formulation for structural mechanical optimization are collected for the comparative evaluation in Schittkowski et al.[37]. A set of 950 least squares test problems solved by an extension of the code NLPQL to retain typical features of a Gauss-Newton algorithm, is described in [35]. Also these problems can be downloaded from the home page of the author together with an interactive user interface called EASY-FIT, see [36].

Moreover, there exist hundreds of commercial and academic applications of NLPQL, for example

1. mechanical structural optimization, see Schittkowski, Zillober, Zotemantel[37] and Kneppe, Krammer, Winkler[16],
2. data fitting and optimal control of transdermal pharmaceutical systems, see Boderke, Schittkowski, Wolf[1] or Blatt, Schittkowski[3],
3. computation of optimal feed rates for tubular reactors, see Birk, Liepelt, Schittkowski, and Vogel[2],
4. food drying in a convection oven, see Frias, Oliveira, and Schittkowski[12],

5. optimal design of horn radiators for satellite communication, see Hartwanger, Schittkowski, and Wolf[11],
6. receptor-ligand binding studies, see Schittkowski[33],
7. optimal design of surface acoustic wave filters for signal processing, see Bünner, Schittkowski, and van de Braak[5].

The general availability of parallel computers and in particular of distributed computing in networks motivates a careful redesign of NLPQL to allow simultaneous function evaluations. The resulting code is called NLPQLP and its mathematical background and some numerical test results are documented in this paper.

The iterative process of an SQP algorithm is highly sequential. Proceeding from a given initial design, new iterates are computed based only on the information from the previous iterate. Each step requires the evaluation of all model functions $f(x_k)$ and $g_j(x_k)$, $j = 1, \ldots, m$, and of gradients $\nabla f(x_k)$ and $\nabla g_j(x_k)$, $j \in J_k$. x_k is the current iterate and $J_k \subset \{1, \ldots, m\}$ a suitable active set determined by the algorithm.

The most effective possibility to exploit a parallel system architecture occurs, when gradients cannot be calculated analytically, but have to be approximated numerically, for example by forward differences, two-sided differences, or even higher order methods. Then we need at least n additional function calls, where n is the number of optimization variables, or a suitable multiple of n. Assuming now that a parallel computing environment is available with l processors, we need only one simultaneous function evaluation in each iteration for calculating gradients, if $l \geq n$. In the most simple case, we have to execute a given simulation program to get $f(x_k + h_{ik}e_i)$ and $g_j(x_k + h_{ik}e_i)$, $j = 1, \ldots, m$, for all $i = 1, \ldots, n$, on n different processors, where h_{ik} is a small perturbation of the i-th unit vector scaled by the actual value of the i-th coefficient of x_k. Subsequently the partial derivatives are approximated by forward differences

$$\frac{1}{h_{ik}}(f(x_k + h_{ik}e_i) - f(x_k)) \ , \quad \frac{1}{h_{ik}}(g_j(x_k + h_{ik}e_i) - g_j(x_k))$$

for $j \in J_k$. Two-sided differences can be used, if $2n \leq l$, fourth-order differences in case of $4n \leq l$, etc.

Another reason for an SQP code to require function evaluations, is the line search. Based on the gradient information at an actual iterate $x_k \in \mathbb{R}^n$, a quadratic programming (QP) problem is formulated and solved to get a search direction $d_k \in \mathbb{R}^n$. It must be ensured that the solution of the QP is a descent direction subject to a certain merit function. Then a sequential line search along $x_k + \alpha d_k$ is performed by combining quadratic interpolation and a steplength reduction. The iteration is stopped as soon as a sufficient descent property is satisfied, leading to a steplength α_k and a new iterate $x_{k+1} = x_k + \alpha_k d_k$. We know that the line search can be restricted to the interval $0 < \alpha \leq 1$, since $\alpha_k = 1$ is expected close to a solution, see e.g. Spellucci[38], because of the local superlinear convergence of an SQP algorithm. Thus, the line search is always started at $\alpha_1 = 1$.

To outline the new approach let us assume that functions can be computed simultaneously on l different machines. Then l test values $\alpha_i = \beta^{i-1}$ with $\beta = \epsilon^{1/(l-1)}$ are selected, $i = 1, \ldots, l$, where ϵ is a guess for the machine precision. Next we require l parallel function calls to get the corresponding model function values. The first α_i satisfying a sufficient descent property, say for $i = i_k$, is accepted as the new steplength for getting the subsequent iterate with $\alpha_k := \alpha_{i_k}$. One has to be sure that existing convergence results of the SQP algorithm are not violated. For an alternative approach based on pattern search, see Hough, Kolda, and Torczon[15].

The paradigm of parallelism is SPMD, i.e., Single Program Multiple Data. In a typical situation we suppose that there is a complex application code providing simulation data, for example by an expensive Finite Element calculation in mechanical structural engineering. It is supposed that various instances of the simulation code providing function values, are executable on a series of different machines, so-called slaves, controlled by a master program that executes NLPQLP. By a message passing system, for example PVM, see Geist et al.[7], only very few data need to be transferred from the master to the slaves. Typically only a set of design parameters of length n must to be passed. On return, the master accepts new model responses for objective function and constraints, at most $m+1$ double precision numbers. All massive numerical calculations and model data, for example the stiffness matrix of a Finite Element model, remain on the slave processors of the distributed system.

The investigations of this paper do not require a special parallel system architecture. We present only a variant of an existing SQP code for nonlinear programming, that can be embedded into an arbitrary distributed environment. A realistic implementation depends highly on available hardware, operating system, or virtual machine, and particularly on the underlying simulation package by which function values are to be computed.

In Section 2 we outline the general mathematical structure of an SQP algorithm, and consider some details of quasi-Newton updates and merit functions in Section 3. Sequential and parallel line search algorithms are described in Section 4. It is shown how the traditional approach is replaced by a more restrictive one with predetermined simultaneous function calls, nevertheless guaranteeing convergence. Numerical results are summarized in Section 5. First it is shown, how the parallel execution of the merit function depends on the number of available machines. Also we compare the results with those obtained by full sequential line search. Since parallel function evaluations are highly valuable in case of numerical gradient computations, we compare also the effect of several difference formulae. Model functions are often disturbed in practical environments, for example in case of iterative algorithms required for internal auxiliary computations. Thus, we add random errors to simulate uncertainties in function evaluations, and compare the overall efficiency of an SQP algorithm. Implementation details of the Fortran subroutine are found in Section 6 together with an illustrative example.

2 SEQUENTIAL QUADRATIC PROGRAMMING METHODS

Sequential quadratic programming or SQP methods belong to the most powerful nonlinear programming algorithms we know today for solving differentiable nonlinear programming problems of the form (1) or (2), respectively. The theoretical background is described e.g. in Stoer[39] in form of a review, or in Spellucci[38] in form of an extensive text book. From the more practical point of view, SQP methods are also introduced in the books of Papalambros, Wilde [18] and Edgar, Himmelblau[6]. Their excellent numerical performance was tested and compared with other methods in Schittkowski[22], and since many years they belong to the most frequently used algorithms to solve practical optimization problems.

The basic idea is to formulate and solve a quadratic programming subproblem in each iteration which is obtained by linearizing the constraints and approximating the Lagrangian function

$$L(x,u) := f(x) - \sum_{j=1}^{m} u_j g_j(x) \tag{3}$$

quadratically, where $x \in I\!R^n$ is the primal variable, and where $u = (u_1, \ldots, u_m)^T \in I\!R^m$ is the multiplier vector. To formulate the quadratic programming subproblem, we proceed from given iterates $x_k \in I\!R^n$, an approximation of the solution, $v_k \in I\!R^m$ an approximation of the multipliers, and $B_k \in I\!R^{n \times n}$, an approximation of the Hessian of the Lagrangian function. Then one has to solve the quadratic programming problem

$$d \in I\!R^n : \begin{array}{l} \min \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d \\ \nabla g_j(x_k)^T d + g_j(x_k) = 0 \ , \quad j = 1, \ldots, m_e \ , \\ \nabla g_j(x_k)^T d + g_j(x_k) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ . \end{array} \tag{4}$$

Let d_k be the optimal solution and u_k be the corresponding multiplier vector of this subproblem. A new iterate is obtained by

$$\begin{pmatrix} x_{k+1} \\ v_{k+1} \end{pmatrix} := \begin{pmatrix} x_k \\ v_k \end{pmatrix} + \alpha_k \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} \ , \tag{5}$$

where $\alpha_k \in (0, 1]$ is a suitable steplength parameter.

The motivation for the success of SQP methods is based on the following observation: **A SQP method is identical to Newton's method to solve the necessary optimality conditions, if B_k is the Hessian of the Lagrangian function and if we start sufficiently close to a solution.** The statement is easily derived in case of equality constraints only, that is $m_e = m$, but holds also for inequality restrictions. A straightforward analysis shows that if $d_k = 0$ is an optimal solution of (4) and u_k the corresponding multiplier vector, then x_k and u_k satisfy the necessary optimality conditions of (2).

Although we are able to guarantee that the matrix B_k is positive definite, it is possible that (4) is not solvable due to inconsistent constraints. One possible remedy is to introduce an additional variable $\delta \in I\!R$, leading to the modified problem

$$
\begin{array}{ll}
 & \min \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d + \sigma_k \delta^2 \\
d \in I\!R^n, & \nabla g_j(x_k)^T d + (1-\delta) g_j(x_k) \left\{ \begin{array}{c} = \\ \geq \end{array} \right\} 0, \quad j \in J_k \ , \\
\delta \in I\!R: & \nabla g_j(x_{k(j)})^T d + g_j(x_k) \geq 0, \qquad\qquad j \in K_k \\
 & 0 \leq \delta \leq 1 \ .
\end{array} \tag{6}
$$

σ_k is a suitable penalty parameter to force that the influence of the additionally introduced variable δ is as small as possible, cf. Schittkowski[27] for details. The active set J_k is given by

$$
J_k := \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, \ g_j(x_k) < \epsilon \quad \text{or} \quad u_j^k > 0\} \tag{7}
$$

and K_k is the complement, i.e. $K_k := \{1, \ldots, m\} \backslash J_k$. ϵ is any small tolerance to define the active constraints, and u_j^k denotes the j-th coefficient of u_k. Obviously, the point $d_0 = 0$, $\delta_0 = 1$ satisfies the linear constraints of (6) which is always solvable. Moreover it is possible to avoid unnecessary gradient evaluations by recalculating only those gradients of restriction functions, that belong to the active set, as indicated by the index '$k(j)$'.

3 MERIT FUNCTIONS AND QUASI-NEWTON UPDATES

The steplength parameter α_k is required in (5) to enforce global convergence of the SQP method, i.e., the approximation of a point satisfying the necessary Karush-Kuhn-Tucker optimality conditions when starting from arbitrary initial values, typically a user-provided $x_0 \in I\!R^n$ and $v_0 = 0$, $B_0 = I$. α_k should satisfy at least a sufficient decrease condition of a merit function $\phi_r(\alpha)$ given by

$$
\phi_r(\alpha) := \psi_r \left(\begin{pmatrix} x \\ v \end{pmatrix} + \alpha \begin{pmatrix} d \\ u - v \end{pmatrix} \right) \tag{8}
$$

with a suitable penalty function $\psi_r(x, v)$. Possible choices of ψ_r are the L_1-penalty function

$$
\psi_r(x, v) := f(x) + \sum_{j=1}^{m_e} r_j |g_j(x)| + \sum_{j=m_e+1}^{m} r_j |\min(0, g_j(x))| \ , \tag{9}
$$

cf. Han[10] and Powell[19], or the augmented Lagrangian function

$$
\psi_r(x, v) := f(x) - \sum_{j \in J} (v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2) - \frac{1}{2} \sum_{j \in K} v_j^2 / r_j \ , \tag{10}
$$

with $J := \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, g_j(x) \leq v_j / r_j\}$ and $K := \{1, \ldots, m\} \setminus J$, cf. Schittkowski[27]. In both cases the objective function is *penalized* as soon as an iterate leaves the feasible domain.

The corresponding penalty parameters that control the degree of constraint violation, must be chosen in a suitable way to guarantee a descent direction of the merit function. Possible choices are

$$r_j^{(k)} := \max\left(|u_j^{(k)}| \ , \ \frac{1}{2}(r_j^{(k-1)} + |u_j^{(k)}|) \right) \ ,$$

see Powell[19] for the L_1-merit function (9), or

$$r_j^{(k)} := \max\left(\frac{2m(u_j^{(k)} - v_j^{(k)})^2}{(1-\delta_k)\ d_k^T B_k d_k} \ , r_j^{(k-1)} \right) \tag{11}$$

for the augmented Lagrangian function (10), see Schittkowski[27]. Here δ_k is the additionally introduced variable to avoid inconsistent quadratic programming problems, see (6). For both merit functions we get the following descent property that is essential to prove convergence.

$$\phi_{r_k}'(0) = \triangledown \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} < 0 \ . \tag{12}$$

For the proof see Han[10] or Schittkowski[27].

Finally one has to approximate the Hessian matrix of the Lagrangian function in a suitable way. To avoid calculation of second derivatives and to obtain a final superlinear convergence rate, the standard approach is to update B_k by the BFGS quasi-Newton formula, cf. Powell[20] or Stoer[39]. The calculation of any new matrix B_{k+1} depends only on B_k and two vectors

$$\begin{aligned} q_k &:= \nabla_x L(x_{k+1}, u_k) - \nabla_x L(x_k, u_k) \ , \\ w_k &:= x_{k+1} - x_k \ , \end{aligned} \tag{13}$$

i.e.,

$$B_{k+1} := \Pi(B_k, q_k, w_k) \ , \tag{14}$$

where

$$\Pi(B, q, w) := B + \frac{qq^T}{q^T w} - \frac{Bww^T B}{w^T B w} \ . \tag{15}$$

The above formula yields a positive definite matrix B_{k+1} provided that B_k is positive definite and $q_k^T w_k > 0$. A simple modification of Powell[19] guarantees positive definite matrices even if the latter condition is violated.

There remains the question whether the convergence of an SQP method can be proved in a mathematically rigorous way. In fact, there exist numerous theoretical convergence results in the literature, see e.g. Spellucci[38]. We want to give here only an impression about the type of these statements, and repeat two results that have been stated in the early days of the SQP methods.

In the first case, we consider the global convergence behaviour, i.e., the question, whether the SQP methods converges when starting from an arbitrary initial point. Suppose that the augmented Lagrangian merit function (8) is implemented and that the primal and dual variables are updated in the form (10).

Theorem 3.1 *Let $\{(x_k, v_k)\}$ be a bounded iteration sequence of the SQP algorithm with a bounded sequence of quasi-Newton matrices $\{B_k\}$ and assume that there are positive constants γ and $\bar{\delta} < 1$ with*

(i) $d_k^T B_k d_k \geq \gamma\, d_k^T d_k$ *for all* k *and a* $\gamma > 0$,

(ii) $\delta_k \leq \bar{\delta}$ *for all* k ,

(iii) $\sigma_k \geq \dfrac{\|A(x_k)\, v_k\|^2}{\gamma(1-\bar{\delta})^2}$ *for all* k .

Then there exists an accumulation point of $\{(x_k, v_k)\}$ satisfying the Karush-Kuhn-Tucker conditions for (2).

Assumption (i) is very well known from unconstrained optimization. It says that the angles between the steepest descent directions and the search directions obtained from the quadratic programming subproblems, must be bounded away from $\pi/2$. Assumptions (ii) and (iii) are a bit more technical and serve to control the additionally introduced variable δ for preventing inconsistency.

The proof of the theorem is found in Schittkowski[27]. The statement is quite weak, but without any further information about second derivatives, we cannot guarantee that the approximated point is indeed a local minimizer.

To investigate now the local convergence speed, we assume that we start from an initial point x_0 sufficiently close to an optimal solution. General assumptions for local convergence analysis are

- $z^\star = (x^\star, u^\star)$ is a strong local minimizer of (2).
- $m_e = m$, i.e., we know all active constraints,
- f, g_1, ..., g_m are twice continuously differentiable,
- for $z_k := (x_k, v_k)$ we have $\lim_{k\to\infty} z_k = z^\star$,
- the gradients $\bigtriangledown g_1(x^\star)$, ..., $\bigtriangledown g_m(x^\star)$ are linearly independent, i.e., the constraint qualification is satisfied,
- $d^T B_k d \geq \gamma\ d^T d$ for all $d \in R^n$ with $A(x_k)^T d = 0$, i.e., the eigenvalues of the Hessian approximations are bounded away from zero.

Powell[20] proved the following theorem for the BFGS update formula.

Theorem 3.2 *Assume that*

(i) $\bigtriangledown_x^2 L(x^\star, u^\star)$ *is positive definite,*

(ii) $\alpha_k = 1$ *for all* k.

Then the sequence $\{x_k\}$ converges R-superlinearly, i.e.,

$$\lim_{k\to\infty} \|x_{k+1} - x^\star\|^{1/k} = 0 \ .$$

The R-superlinear convergence speed is somewhat weaker than the Q-superlinear convergence rate defined below. It was Han[9] who proved the statement

$$\lim_{k\to\infty} \frac{\|z_{k+1} - z^\star\|}{\|z_k - z^\star\|} = 0 \ .$$

for the so-called DFP update formula, a slightly different quasi-Newton method. In this case, we get a sequence β_k tending to zero with

$$\|z_{k+1} - z^\star\| \le \beta_k \, \|z_k - z^\star\|$$

4 STEPLENGTH CALCULATION

Let us consider in more detail, how a steplength α_k is actually calculated. First we select a suitable merit function, in our case the augmented Lagrangian (10), that defines a scalar function $\phi_r(\alpha)$. For obvious reasons, a full minimization along α is not possible. The idea is to get a sufficient decrease for example measured by the so-called Goldstein condition

$$\phi_r(0) + \alpha\mu_2\phi_r'(0) \le \phi_r(\alpha) \le \phi_r(0) + \alpha\mu_1\phi_r'(0) \tag{16}$$

or the Armijo condition

$$\phi_r(\sigma\beta^i) \le \phi_r(0) + \sigma\beta^i\mu\phi_r'(0) \ , \tag{17}$$

see for example Ortega and Rheinboldt[17]. The constants are from the ranges $0 < \mu_1 \le 0.5 < \mu_2 < 1$, $0 < \mu < 0.5$, $0 < \beta < 1$, and $0 < \sigma \le 1$. In the first case, we accept any α in the range given by (16), whereas the second condition is constructive. We start with $i = 0$ and increase i until (17) is satisfied for the first time, say at i_k. Then the desired steplength is $\alpha_k = \sigma\beta^{i_k}$. Both approaches are feasible because of the descent property $\phi_r'(0) < 0$, see (12).

All line search algorithms have to satisfy two requirements, which are somewhat contradicting:

1. The decrease of the merit function must be sufficiently large, to accelerate convergence.
2. The steplength must not become too small to avoid convergence against a non-stationary point.

The implementation of a line search algorithm is a critical issue when implementing a nonlinear programming algorithm, and has significant effect on the overall efficiency of the resulting code. On the one hand we need a line search to stabilize the algorithm, on the other hand it is not advisable to waste too many function calls. Moreover, the behavior of the merit function becomes irregular in case of constrained optimization problems because of very steep slopes at the border caused by the penalty terms. Even the implementation is more complex than shown above, if linear constraints and bounds of the variables are to be satisfied during the line search.

Fortunately, SQP methods are quite robust and accept the steplength one in the neighborhood of a solution. Typically the test parameter μ for the Armijo-type sufficient descent property (17) is very small, for example $\mu = 0.0001$ in the present implementation of NLPQL. Nevertheless the choice of the reduction parameter β must be adopted to the actual slope of the merit function. If β is too small, the line search terminates very fast, but on the other hand the resulting stepsizes are usually too small leading to a higher number of outer iterations. On the other hand, a larger value close to one requires too many function calls during the line search. Thus, we need some kind of compromise, which is obtained by applying first a polynomial interpolation, typically a quadratic one, and use (16) or (17) only as a stopping criterion. Since $\phi_r(0)$, $\phi_r'(0)$, and $\phi_r(\alpha_i)$ are given, α_i the actual iterate of the line search procedure, we get easily the minimizer of the quadratic interpolation. We accept then the maximum of this value or the Armijo parameter as a new iterate, as shown by the subsequent code fragment implemented in NLPQL.

Algorithm 4.1 *Let* β, μ *with* $0 < \beta < 1$, $0 < \mu < 0.5$ *be given.*

Start: $\alpha_0 := 1$

For $i = 0, 1, 2, \ldots$ *do:*

1) *If* $\phi_r(\alpha_i) < \phi_r(0) + \mu\, \alpha_i\, \phi_r'(0)$, *then stop.*

2) *Compute* $\bar{\alpha}_i := \dfrac{0.5\, \alpha_i^2\, \phi_r'(0)}{\alpha_i \phi_r'(0) - \phi_r(\alpha_i) + \phi_r(0)}$.

3) *Let* $\alpha_{i+1} := \max(\beta\, \alpha_i, \bar{\alpha}_i)$.

Corresponding convergence results are found in Schittkowski[27]. $\bar{\alpha}_i$ is the minimizer of the quadratic interpolation and we use the Armijo descent property for checking termination. Step 3) is required to avoid irregular values, since the minimizer of the quadratic interpolation could be outside of the feasible domain $(0, 1]$. The search algorithm is implemented in NLPQL together with additional safeguards, for example to prevent violation of bounds. Algorithm 4.1 assumes that $\phi_r(1)$ is known before calling the procedure, i.e., the corresponding function call is made in the calling program. We have to stop the algorithm, if sufficient descent is not observed after a certain number of iterations, say 10. If the tested stepsizes fall below machine precision or the accuracy by which model function values are computed, the merit function cannot decrease further.

Now we come back to the question, how the sequential line search algorithm can be modified to work under a parallel computing environment. Proceeding from an existing implementation as outlined above, the answer is quite simple. To outline the new approach, let us assume that functions can be computed simultaneously on l different machines. Then l test values $\alpha_i = \beta^i$ with $\beta = \epsilon^{1/(l-1)}$ are selected, $i = 0, \ldots, l-1$, where ϵ is a guess for the machine precision. Next we order l parallel function calls to get $f(x_k + \alpha_i d_k)$ and $g_j(x_k + \alpha_i d_k)$, $j = 1, \ldots, m$, for $i = 0, \ldots, l-1$. The first α_i satisfying the sufficient descent property (17) at $i = i_k$,

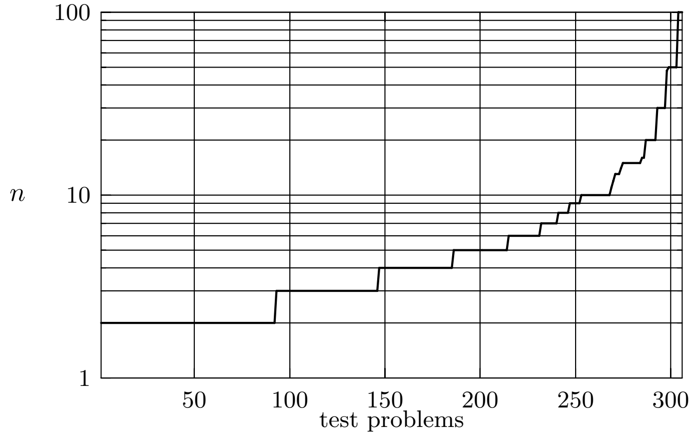

FIGURE 1: Number of Variables

is accepted as the steplength $\alpha_k := \alpha_{i_k}$ for getting the subsequent iterate x_{k+1}.

The proposed parallel line search will work efficiently, if the number of parallel machines l is sufficiently large, and works as follows.

Algorithm 4.2 *Let β, μ with $0 < \beta < 1$, $0 < \mu < 0.5$ be given.*

Start: For $\alpha_i = \beta^i$ compute $\phi_r(\alpha_i)$ for $i = 0, \ldots, l-1$.

For $i = 0, 1, 2, \ldots$ do:

If $\phi_r(\alpha_i) < \phi_r(0) + \mu\ \alpha_i\ \phi_r'(0)$, then stop.

To precalculate l candidates in parallel at log-distributed points between a small tolerance $\alpha = \tau$ and $\alpha = 1$, $0 < \tau << 1$, we propose $\beta = \tau^{1/(l-1)}$.

5 NUMERICAL RESULTS

Our numerical tests use all 306 academic and real-life test problems published in Hock and Schittkowski[14] and in Schittkowski[30]. The Fortran source codes can be downloaded from the home page of the author

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

together with a user's guide, see Schittkowski[34]. The distribution of the dimension parameter n, the number of variables, is shown in Figure 1. We see, for example, that about 270 of 306 test problems have not more than 10 variables. In a similar way, the distribution of the number of constraints is shown in Figure 2.

Since analytical derivatives are not available for all problems, we approximate them numerically. The test examples are provided with exact solutions, either known from analytical solutions or from the best numerical data found so far. The Fortran code is compiled by the Compaq Visual Fortran Optimizing Compiler,

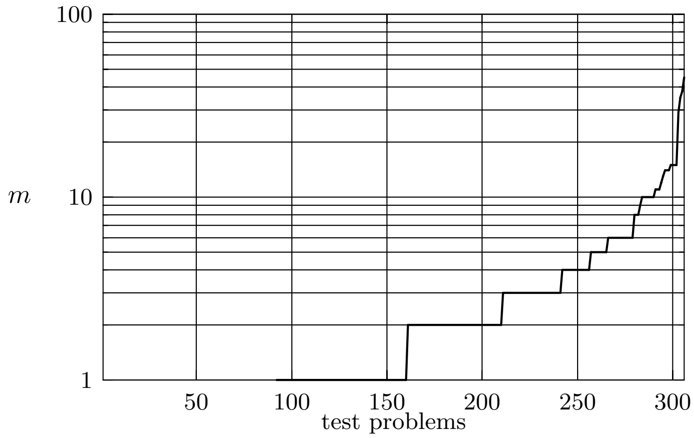

FIGURE 2: Number of Constraints

Version 6.5, under Windows 2000, and executed on a Pentium III processor with 750 MHz. Since the calculation times are very short, about 10 sec for solving all 306 test problems, we count only function and gradient evaluations. This is a realistic assumption, since for the practical applications we have in mind, calculation times for evaluating model functions dominate and the numerical efforts within NLPQLP are negligible.

First we need a criterion to decide, whether the result of a test run is considered as a successful return or not. Let $\epsilon > 0$ be a tolerance for defining the relative termination accuracy, x_k the final iterate of a test run, and $x^\star$ the supposed exact solution as reported by the two test problem collections. Then we call the output of an execution of NLPQLP a successful return, if the relative error in objective function is less than ϵ and if the sum of all constraint violations less than ϵ^2, i.e., if

$$f(x_k) - f(x^\star) < \epsilon |f(x^\star)| \ , \ \text{if } f(x^\star) <> 0 \ \ ,$$

or

$$f(x_k) < \epsilon \ , \ \text{if } f(x^\star) = 0 \ \ ,$$

and

$$r(x_k) := \sum_{j=1}^{m_e} |g_j(x_k)| + \sum_{j=m_e+1}^{m} |\min(0, g_j(x_k))| < \epsilon^2 \ \ .$$

We take into account that NLPQLP returns a solution with a better function value than the known one, subject to the error tolerance of the allowed constraint violation. However there is still the possibility that NLPQLP terminates at a local solution different from the one known in advance. Thus, we call a test run a successful one, if NLPQLP terminates with error message IFAIL=0, and if

$$f(x_k) - f(x^\star) \geq \epsilon |f(x^\star)| \ , \ \text{if } f(x^\star) <> 0 \ \ ,$$

L	SUCC	NF	NIT
1	306	41	25
3	206	709	179
4	251	624	126
5	282	470	80
6	291	339	50
7	292	323	42
8	297	299	35
9	299	305	32
10	300	300	29
12	301	346	28
15	297	394	26
20	299	519	26
50	300	1,280	26

TABLE 1: Performance Results for Parallel Line Search

or

$$f(x_k) \geq \epsilon \ , \ \text{if } f(x^\star) = 0 \ \ ,$$

and

$$r(x_k) < \epsilon^2 \ \ .$$

For our numerical tests, we use $\epsilon = 0.01$, i.e., we require a final accuracy of one per cent. NLPQLP is executed with termination accuracy ACC$=10^{-8}$, and MAXIT=500. Termination with IFAIL=0 indicates that all internal stopping criteria are satisfied. Gradients are approximated by a fourth-order difference formula

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{4!\eta_i}\left(2f(x-2\eta_i e_i) - 16 f(x-\eta_i e_i) + 16 f(x+\eta_i e_i) - 2f(x+2\eta_i e_i) \right) \ \ , \quad (18)$$

where $\eta_i = \eta \max(10^{-5}, |x_i|)$, $\eta = 10^{-7}$, e_i the i-th unit vector, and $i = 1, \ldots, n$. In a similar way, derivatives of the constraint functions are computed.

First we investigate the question, how parallel line searches influence the overall performance. Table 1 shows the number of successful test runs SUCC, the average number of function calls NF, and the average number of iterations NIT, for increasing number of simulated parallel calls of model functions denoted by L. To get NF, we count each single function call, also in the case L>1. However, function evaluations needed for gradient approximations, are not counted. Their average number is 4×NIT.

L=1 corresponds to the sequential case, when Algorithm 4.1 is applied for the line search, consisting of a quadratic interpolation combined with an Armijo-type bisection strategy. In this case, all problems can be solved successfully. Since we need at least one function evaluation for the subsequent iterate, we observe that the average number of additional function evaluations needed for the line search, is less than one.

In all other cases, L>1 simultaneous function evaluations are made according to Algorithm 4.2. Thus, the total number of function calls NF is quite big in Table 1. If, however, the number of parallel machines L is sufficiently large in a practical situation, we need only one simultaneous function evaluation in each step of the SQP algorithm. To get a reliable and robust line search, we need at least 5 parallel processors. No significant improvements are observed, if we have more than 10 parallel function evaluations.

The most promising possibility to exploit a parallel system architecture occurs, when gradients cannot be calculated analytically, but have to be approximated numerically, for example by forward differences, two-sided differences, or even higher order methods. Then we need at least n additional function calls, where n is the number of optimization variables, or a suitable multiple of n.

For our numerical tests, we implement three different approximation routines for derivatives based on standard difference formulae of increasing order:

1. Forward differences:

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{\eta_i}\Big(f(x+\eta_i e_i) - f(x) \Big)$$

2. Two-sided differences:

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{2\eta_i}\Big(f(x+\eta_i e_i) - f(x-\eta_i e_i \Big)$$

3. Fourth-order formula:

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{4!\eta_i}\Big(2f(x-2\eta_i e_i) - 16f(x-\eta_i e_i) + 16f(x+\eta_i e_i) - 2f(x+2\eta_i e_i) \Big)$$

In the above formulae, $i = 1, \ldots, n$ is the index of the variables for which a partial derivative is to be computed, $x = (x_1, \ldots, x_n)^T$ the argument, e_i the i-the unit vector, and $\eta_i = \eta \max(10^{-5}, |x_i|)$ the relative perturbation. In the same way, derivatives for constraints are approximated.

Table 2 shows the corresponding results for the different procedures under consideration, and for increasing random perturbations (ERR). We report the number of successful runs (SUCC) only, since the average number of iterations is more or less the same in all cases. The tolerance for approximating gradients, η, is set to the square root of ERR, and the termination tolerance of NLPQLP is set to ACC=0.1×ERR.

6 IMPLEMENTATION DETAILS

NLPQLP is implemented in form of a Fortran subroutine to solve nonlinear programs of the form (1). The quadratic programming problem is solved by the code QL, an implementation of the primal-dual method of Goldfarb and Idnani[8] going

ERR	1	2	3
0	298	298	306
10^{-12}	292	297	297
10^{-10}	267	284	277
10^{-8}	236	254	255
10^{-6}	207	231	221
10^{-4}	137	175	171

TABLE 2: Successful Test Runs for Different Gradient Approximations

back to Powell[21]. Model functions and gradients are called by reverse communication. The code is executed with the calling sequence

```
      CALL   NLPQLP(L,M,ME,MMAX,N,NMAX,MNN2,X,F,G,DF,DG,U,XL,
     /              XU,C,D,ACC,ACCQP,STPMIN,MAXFUN,MAXIT,IPRINT,
     /              MODE,IOUT,IFAIL,WA,LWA,KWA,LKWA,ACT,LACT)
```

Most of the parameters are only needed to pass variable, function, gradient and multiplier values. Moreover, real, integer, and logical working arrays are required. NLPQLP is a very robust implementation of a sequential quadratic programming algorithm and requires very few user-provided parameters besides of the organizational ones. Only two input parameters are essential,

MAXIT	:	maximum number of iterations,
ACC	:	final termination accuracy.

Any reasonably large number of iterations MAXIT can be used, since we suppose that NLPQLP is executed to solve small-scale, but eventually highly nonlinear optimization problems. However, if function and gradient evaluations provided by the simulation code, are expensive, it is recommended to perform only a few iterations and to perform a restart if the final iterate is not acceptable.

Several termination criteria are implemented in NLPQLP to prevent scaling effects as much as possible, and to stop the algorithm as soon as the desired tolerance ACC is reached. For more details, see Schittkowski[29]. If gradients are evaluated within machine precision, a relatively small parameter between 1.0E-8 to 1.0E-14 is acceptable. Otherwise, ACC should not be smaller than the stepsize applied for a numerical gradient approximation by forward differences, or the accuracy of objective function and constraint evaluation.

All other parameters, solution tolerances, or options usually required by a nonlinear programming code are set internally. The parameter MODE specifies the desired version of NLPQLP. For MODE=0, initial guesses for multipliers and the Hessian approximation are set to the zero vector and the unit matrix, respectively. MODE=1 indicates that the user wants to provide an initial guesses and to store them in U and C, respectively. The tolerance ACCQP 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 zero, the machine precision is computed by NLPQLP and subsequently multiplied by 10^4. The minimum steplength STPMIN is used in case of L>1. Recommended is any value in the order of the accuracy by which functions are computed. The value is needed to compute a steplength reduction factor by STPMIN**(1/(L-1)). If STPMIN=0, then STPMIN=ACC is used.

The following termination messages are displayed by NLPQLP:

IFAIL = 0	:	Optimality conditions are satisfied.
IFAIL = 1	:	The algorithm terminates after MAXIT iterations.
IFAIL = 2	:	The algorithm computes an uphill search direction.
IFAIL = 3	:	Underflow occurs when determining a new approximation of the Hessian matrix of the Lagrangian function.
IFAIL = 4	:	The line search is not terminated successfully.
IFAIL = 7	:	The search direction is close to zero, but the current iterate is still infeasible.
IFAIL>10	:	The quadratic programming code terminates with an error message IFQL>0 and IFAIL is set to IFAIL=IFQL+10.

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, there the correctness of the model must be checked very carefully.
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 require 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 the optimal solution. In this situation, it is recommended to check the formulation of the model.

However some of the error situations do also occur, if because of wrong or non-accurate 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.

The user has to provide functions and gradients in the same program, which executes NLPQLP, according to the following rules:

1. Choose starting values for the variables to be optimized, and store them in the first column of X.
2. Compute objective and all constraint function values, store them in F(1) and the first column of G, respectively.
3. 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 J-th constraint, J=1,...,M.
4. Set IFAIL=0 and execute NLPQLP.
5. If NLPQLP returns with IFAIL=-1, compute objective function and constraint values for all variables found in the first L columns of X, store them in F (first L positions) and G (first L columns), and call NLPQLP again.
6. If NLPQLP terminates with IFAIL=-2, compute gradient values with respect to the variables stored in the first column of X, and store them in DF and DG. Only derivatives for active constraints, ACT(J)=.TRUE., need to be computed. Then call NLPQLP again.
7. If NLPQLP terminates with IFAIL=0, the internal stopping criteria are satisfied. In case of IFAIL>0, an error occurred.

The above strategy for providing function and gradient values, is called reverse communication. If analytical derivatives are not available, simultaneous function calls can be used for gradient approximations, for example by forward differences (2N>L), two-sided differences (4N>L≥2N), or even higher order formulae (L≥4N).

To give an example how to organize the code, we consider Rosenbrock's post office problem, i.e., test problem TP37 of Hock and Schittkowski[14].

$$x_1, x_2 \in I\!R : \begin{array}{l} \min -x_1 x_2 x_3 \\ x_1 + 2x_2 + 2x_3 \geq 0 \\ 72 - x_1 - 2x_2 - 2x_3 \geq 0 \\ 0 \leq x_1 \leq 100 \\ 0 \leq x_2 \leq 100 \end{array} \tag{19}$$

The Fortran source code for executing NLPQLP is listed below. Gradients are approximated by forward differences. The function block inserted in the main program can be replaced by a subroutine call. Also the gradient evaluation is easily exchanged by an analytical one or higher order derivatives.

```
      IMPLICIT NONE
      INTEGER   NMAX,MMAX,LMAX,MNN2X,LWA,LKWA,LACT
      PARAMETER (NMAX=4,MMAX=2,LMAX=10)
      PARAMETER (MNN2X = MMAX+NMAX+NMAX+2,
     /          LWA=3*NMAX*NMAX/2+6*MMAX+28*NMAX+100,
     /          LKWA=MMAX+2*NMAX+20,LACT=2*MMAX+15)
      INTEGER   KWA(LKWA),N,ME,M,L,MNN2,MAXIT,MAXFUN,IPRINT,
     /          IOUT,MODE,IFAIL,I,J,K,NFUNC
      DOUBLE PRECISION X(NMAX,LMAX),F(LMAX),G(MMAX,LMAX),DF(NMAX),
     /          DG(MMAX,NMAX),U(MNN2X),XL(NMAX),XU(NMAX),C(NMAX,NMAX),
     /          D(NMAX),WA(LWA),ACC,ACCQP,STPMIN,EPS,EPSREL,FBCK,
     /          GBCK(MMAX),XBCK
      LOGICAL   ACT(LACT)
```

```
      IOUT=6
      ACC=1.0D-9
      ACCQP=1.0D-11
      STPMIN=1.0E-10
      EPS=1.0D-7
      MAXIT=100
      MAXFUN=10
      IPRINT=2
      N=3
      L=N
      M=2
      ME=0
      MNN2=M+N+N+2
      DO I=1,N
         DO K=1,L
            X(I,K)=1.0D+1
         ENDDO
         XL(I)=0.0
         XU(I)=1.0D+2
      ENDDO
      MODE=0
      IFAIL=0
      NFUNC=0
    1 CONTINUE
C============================================================
C   This is the main block to compute all function values
C   simultaneously, assuming that there are L nodes.
C   The block is executed either for computing a steplength
C   or for approximating gradients by forward differences.
      DO K=1,L
         F(K)=-X(1,K)*X(2,K)*X(3,K)
         G(1,K)=X(1,K) + 2.0*X(2,K) + 2.0*X(3,K)
         G(2,K)=72.0 - X(1,K) - 2.0*X(2,K) - 2.0*X(3,K)
      ENDDO
C============================================================
      NFUNC=NFUNC+1
      IF (IFAIL.EQ.-1) GOTO 4
      IF (NFUNC.GT.1) GOTO 3
    2 CONTINUE
      FBCK=F(1)
      DO J=1,M
         GBCK(J)=G(J,1)
      ENDDO
      XBCK=X(1,1)
      DO I=1,N
         EPSREL=EPS*DMAX1(1.0D0,DABS(X(I,1)))
         DO K=2,L
            X(I,K)=X(I,1)
         ENDDO
         X(I,I)=X(I,1)+EPSREL
      ENDDO
      GOTO 1
    3 CONTINUE
      X(1,1)=XBCK
      DO I=1,N
         EPSREL=EPS*DMAX1(1.0D0,DABS(X(I,1)))
         DF(I)=(F(I)-FBCK)/EPSREL
         DO J=1,M
            DG(J,I)=(G(J,I)-GBCK(J))/EPSREL
         ENDDO
      ENDDO
      F(1)=FBCK
      DO J=1,M
         G(J,1)=GBCK(J)
      ENDDO
    4 CALL NLPQLP(L,M,ME,MMAX,N,NMAX,MNN2,X,F,G,DF,DG,U,XL,XU,
     /     C,D,ACC,ACCQP,STPMIN,MAXFUN,MAXIT,IPRINT,MODE,IOUT,
```

```
     /     IFAIL,WA,LWA,KWA,LKWA,ACT,LACT)
      IF (IFAIL.EQ.-1) GOTO 1
      IF (IFAIL.EQ.-2) GOTO 2
      WRITE(IOUT,1000) NFUNC
 1000 FORMAT('   *** Number of function calls: ',I3)
      STOP
      END
```

When applying simultaneous function evaluations with L=N, only 20 function calls and 10 iterations are required to get a solution within termination accuracy 10^{-10}. A call with $L = 1$ would stop after 9 iterations. The following output should appear on screen:

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

  Parameters:
     MODE =  0
     ACC =   0.1000D-08
     MAXFUN =  3
     MAXIT =  100
     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 -0.10000000D+04  0.00D+00   2  0  0.00D+00  0.00D+00  0.46D+04
    2 -0.10003444D+04  0.00D+00   1  2  0.10D-03  0.00D+00  0.38D+04
    3 -0.33594686D+04  0.00D+00   1  1  0.10D+01  0.00D+00  0.24D+02
    4 -0.33818566D+04  0.16D-09   1  1  0.10D+01  0.00D+00  0.93D+02
    5 -0.34442871D+04  0.51D-08   1  1  0.10D+01  0.00D+00  0.26D+03
    6 -0.34443130D+04  0.51D-08   1  2  0.10D-03  0.00D+00  0.25D+02
    7 -0.34558588D+04  0.19D-08   1  1  0.10D+01  0.00D+00  0.30D+00
    8 -0.34559997D+04  0.00D+00   1  1  0.10D+01  0.00D+00  0.61D-03
    9 -0.34560000D+04  0.00D+00   1  1  0.10D+01  0.00D+00  0.12D-07
   10 -0.34560000D+04  0.00D+00   1  1  0.10D+01  0.00D+00  0.13D-10

   --- Final Convergence Analysis ---

     Objective function value:  F(X) = -0.34560000D+04
     Approximation of solution:  X =
        0.24000000D+02  0.12000000D+02  0.12000000D+02
     Approximation of multipliers:  U =
        0.00000000D+00  0.14400000D+03  0.00000000D+00  0.00000000D+00
        0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     Constraint values:  G(X) =
        0.72000000D+02  0.35527137D-13
     Distance from lower bound:  XL-X =
       -0.24000000D+02 -0.12000000D+02 -0.12000000D+02
     Distance from upper bound:  XU-X =
        0.76000000D+02  0.88000000D+02  0.88000000D+02
     Number of function calls:     NFUNC =  10
     Number of gradient calls:     NGRAD =  10
     Number of calls of QP solver: NQL   =  10
```

```
*** Number of function calls:  20
```

In case of L=1, NLPQLP is identical to NLPQL and stops after 9 iterations. The corresponding function block of the main program is much simpler now,

```
C============================================================
C   This is the main block to compute all function values.
C  The block is executed either for computing a steplength
C   or for approximating gradients by a difference formula.
      F(1)=-X(1)*X(2)*X(3)
      G(1)=X(1) + 2.0*X(2) + 2.0*X(3)
      G(2)=72.0 - X(1) - 2.0*X(2) - 2.0*X(3)
C============================================================
```

7 CONCLUSIONS

We present a modification of an SQP algorithm designed for execution under a parallel computing environment (SPMD). Under the assumption that objective functions and constraints are executed on different machines, a parallel line search procedure is proposed. Thus, the SQP algorithm is executable under a distributed system, where parallel function calls are exploited for line search and gradient approximations.

The approach is outlined, some implementation details are presented, and numerical tests are performed. It is shown that there are no significant performance differences between sequential and parallel line searches, if the number of parallel processors is sufficiently large. In both cases, about the same number of iterations is performed, and the number of successfully solved problems is also comparable. The test results are obtained by a collection of 306 academic and real-life examples.

By a series of further tests, it is shown how the code behaves for different gradient approximations under additional random noise added to the model functions, to simulate realistic situations arising in practical applications. If a sufficiently large number of parallel processors is available, it is recommended to apply higher order approximation formulae instead of forward differences.

REFERENCES

1. Boderke P., Schittkowski K., Wolf M., Merkle H.P. (2000): *Modeling of diffusion and concurrent metabolism in cutaneous tissue,* Journal on Theoretical Biology, Vol. 204, No. 3, 393-407
2. Birk J., Liepelt M., Schittkowski K., Vogel F. (1999): *Computation of optimal feed rates and operation intervals for tubular reactors,* Journal of Process Control, Vol. 9, 325-336
3. Blatt M., Schittkowski K. (1998): *Optimal Control of One-Dimensional Partial Differential Equations Applied to Transdermal Diffusion of Substrates*, in: Optimization Techniques and Applications, L. Caccetta, K.L. Teo, P.F. Siew, Y.H. Leung, L.S. Jennings, V. Rehbock eds., School of Mathematics and Statistics, Curtin University of Technology, Perth, Australia, Vol. 1, 81 - 93
4. Bongartz I., Conn A.R., Gould N., Toint Ph. (1995): *CUTE: Constrained and unconstrained testing environment*, Transactions on Mathematical Software, Vol. 21, No. 1, 123-160

5. Bünner M.J., Schittkowski K., van de Braak G. (2002): *Optimal design of surface acoustic wave filters for signal processing by mixed-integer nonlinear programming*, submitted for publication
6. Edgar T.F., Himmelblau D.M. (1988): *Optimization of Chemical Processes,* McGraw Hill
7. Geist A., Beguelin A., Dongarra J.J., Jiang W., Manchek R., Sunderam V. (1995): *PVM 3.0. A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press
8. Goldfarb D., Idnani A. (1983): *A numerically stable method for solving strictly convex quadratic programs*, Mathematical Programming, Vol. 27, 1-33
9. Han S.-P. (1976): *Superlinearly convergent variable metric algorithms for general nonlinear programming problems* Mathematical Programming, Vol. 11, 263-282
10. Han S.-P. (1977): *A globally convergent method for nonlinear programming* Journal of Optimization Theory and Applications, Vol. 22, 297–309
11. Hartwanger C., Schittkowski K., Wolf H. (2000): *Computer aided optimal design of horn radiators for satellite communication*, Engineering Optimization, Vol. 33, 221-244
12. Frias J.M., Oliveira J.C, Schittkowski K. (2001): *Modelling of maltodextrin DE12 drying process in a convection oven*, to appear: Applied Mathematical Modelling
13. Hock W., Schittkowski K. (1981): *Test Examples for Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 187, Springer
14. Hock W., Schittkowski K. (1983): *A comparative performance evaluation of 27 nonlinear programming codes*, Computing, Vol. 30, 335-358
15. Hough P.D., Kolda T.G., Torczon V.J. (2001): *Asynchronous parallel pattern search for nonlinear optimization*, to appear: SIAM J. Scientific Computing
16. Kneppe G., Krammer J., Winkler E. (1987): *Structural optimization of large scale problems using MBB-LAGRANGE*, Report MBB-S-PUB-305, Messerschmitt-Bölkow-Blohm, Munich
17. Ortega J.M., Rheinbold W.C. (1970): *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York-San Francisco-London
18. Papalambros P.Y., Wilde D.J. (1988): *Principles of Optimal Design,* Cambridge University Press
19. 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
20. Powell M.J.D. (1978): *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, in: Nonlinear Programming 3, O.L. Mangasarian, R.R. Meyer, S.M. Robinson eds., Academic Press
21. Powell M.J.D. (1983): *On the quadratic programming algorithm of Goldfarb and Idnani.* Report DAMTP 1983/Na 19, University of Cambridge, Cambridge
22. Schittkowski K. (1980): *Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 183 Springer
23. Schittkowski K. (1981): *The nonlinear programming method of Wilson, Han and Powell. Part 1: Convergence analysis*, Numerische Mathematik, Vol. 38, 83-114
24. Schittkowski K. (1981): *The nonlinear programming method of Wilson, Han and Powell. Part 2: An efficient implementation with linear least squares subproblems*, Numerische Mathematik, Vol. 38, 115-127
25. Schittkowski K. (1982): *Nonlinear programming methods with linear least squares subproblems*, in: Evaluating Mathematical Programming Techniques, J.M. Mulvey ed., Lecture Notes in Economics and Mathematical Systems, Vol. 199, Springer
26. Schittkowski K. (1983): *Theory, implementation and test of a nonlinear programming algorithm*, in: Optimization Methods in Structural Design, H. Eschenauer, N. Olhoff eds., Wissenschaftsverlag
27. Schittkowski K. (1983): *On the convergence of a sequential quadratic programming method with an augmented Lagrangian search direction,* Mathematische Operationsforschung und Statistik, Series Optimization, Vol. 14, 197-216

28. Schittkowski K. (1985): *On the global convergence of nonlinear programming algorithms*, ASME Journal of Mechanics, Transmissions, and Automation in Design, Vol. 107, 454-458
29. Schittkowski K. (1985/86): *NLPQL: A Fortran subroutine solving constrained nonlinear programming problems,* Annals of Operations Research, Vol. 5, 485-500
30. Schittkowski K. (1987a): *More Test Examples for Nonlinear Programming,* Lecture Notes in Economics and Mathematical Systems, Vol. 182, Springer
31. Schittkowski K. (1988): *Solving nonlinear least squares problems by a general purpose SQP-method,* in: Trends in Mathematical Optimization, K.-H. Hoffmann, J.-B. Hiriart-Urruty, C. Lemarechal, J. Zowe eds., International Series of Numerical Mathematics, Vol. 84, Birkhäuser, 295-309
32. Schittkowski K. (1992): *Solving nonlinear programming problems with very many constraints*, Optimization, Vol. 25, 179-196
33. Schittkowski K. (1994): *Parameter estimation in systems of nonlinear equations*, Numerische Mathematik, Vol. 68, 129-142
34. Schittkowski K. (2002): *Test problems for nonlinear programming - user's guide*, Report, Department of Mathematics, University of Bayreuth
35. Schittkowski K. (2002): *Numerical Data Fitting in Dynamical Systems*, to appear: Kluwer Academic Publishers
36. Schittkowski K. (2002): *EASY-FIT: A software system for data fitting in dynamic systems*, to appear: Journal of Design Optimization
37. Schittkowski K., Zillober C., Zotemantel R. (1994): *Numerical comparison of nonlinear programming algorithms for structural optimization*, Structural Optimization, Vol. 7, No. 1, 1-28
38. Spellucci P. (1993): *Numerische Verfahren der nichtlinearen Optimierung,* Birkhäuser
39. Stoer J. (1985): *Foundations of recursive quadratic programming methods for solving nonlinear programs,* in: Computational Mathematical Programming, K. Schittkowski, ed., NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 15, Springer