NLPQLG: A Fortran Implementation of a Sequential Quadratic Programming Algorithm for Heuristic Global Optimization

- User's Guide -

Address:	Prof. K. Schittkowski Siedlerstr. 3 D - 95488 Eckersdorf Germany
Phone:	(+49) 921 32887
E-mail:	klaus@schittkowski.de
Web:	http://www.klaus-schittkowski.de
Date:	December, 2012

Abstract

Usually, global optimization codes with guaranteed convergence require a large number of function evaluations. On the other hand, there are efficient optimization methods which exploit gradient information, but only the approximation of a local minimizer can be expected. If, however, the underlying application model is expensive, if there are additional constraints, especially equality constraints, and if the existence of different local solutions is expected, then heuristic rules for successive switches from one local minimizer to another are often the only applicable approach. For this specific situation, we present some simple ideas for cutting off a local minimizer and to restart a new local optimization run. However, some safeguards are needed to stabilize the algorithm, since very little is usually known about the distribution of local minima. The paper introduces an approach where the nonlinear programs generated can be solved by any available *black box* software. For our implementation, a sequential quadratic programming code (NLPQLP) is chosen for local optimization. The usage of the code is outlined and we present some numerical results based on a set of test examples found in the literature.

Keywords: global optimization, deterministic methods, SQP, sequential quadratic programming, nonlinear programming, numerical algorithm, Fortran codes, test examples

1 Introduction

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

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

where x is an n-dimensional parameter vector. It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on the whole $I\!\!R^n$. But besides of this we do not impose any further restrictions on the mathematical structure.

Let P denote the feasible domain,

$$P := \{ x \in I\!\!R^n : g_j(x) = 0, \ j = 1, \ldots, m_e, \ \ g_j(x) \geq 0, \ j = m_e + 1, \ldots, m, \ \ x_l \leq x \leq x_u \} \ .$$

Our special interest is to find a global optimizer, i.e., a feasible point $x^\star \in P$ with $f(x^\star) \leq f(x)$ for all $x \in P$. Without further assumptions, it is not possible to know in advance how many local solutions exist or even whether the number of local solutions is finite or not, or whether to global minimizer is unique.

Global optimization algorithms have been investigated in the literature extensively, see for example the books of Pinter [19], Törn and Zilinskas [32], Horst and Pardalos [14] and the references herein. Main solution strategies are partition techniques, stochastic algorithms, or approximation techniques, among many other methods. Despite of significant progress and improvements in developing new computer codes, there remains the disadvantage that the number of function evaluations is often large and unacceptable for realistic time-consuming simulations. Especially nonlinear equality constraints are often not appropriate and must be handled in form of penalty terms, by which direct and random search algorithms can be deteriorated drastically.

One of the main drawbacks of global optimization is the lack of numerically computable and generally applicable stopping criteria. Thus, global optimization is inherently a difficult problem and requires more or less an exhaustive search over the whole feasible domain to guarantee convergence.

As soon as function evaluations become extremely expensive preventing the application of any of the methods mentioned above, there are only two alternatives. Either the mathematical model allows specific analysis to restrict the domain of interest to a region where the global minimizer is expected, or one tries to improve local solutions until a *reasonable*, not necessarily global solution is found. A typical technique is the so-called tunnelling method, see Levy and Montalvo [16], where the objective function in (1) is replaced by

$$\overline{f}(x) = \frac{f(x) - f(x^{loc})}{||x - x^{loc}||^\rho}$$

and where $x^{loc} \in P$ denotes a local minimizer. ρ is a penalty parameter with the intention to *push away* the next local minimizer from the known one, x^{loc}. A similar idea to move to another local minimizer, is proposed by Ge and Qin [6], also called function filling method, where the objective function is inverted and an exponential penalty factor added to prevent approximation of a known local solution.

Our heuristic proposal follows a similar idea, the successive improvement of global minima. But the algorithm is different in the following sense. Additional constraints are attached to the original problem formulation (1). First, there is a constraint that the next local solution must have an objective function value less than the best known feasible function value minus a relative improvement of $\epsilon_1 > 0$. For each known local solution x^{loc}, a ball is formulated around x^{loc} with radius ϵ_2 to prevent a subsequent approximation of x^{loc}. The procedure is repeated until the solution method breaks down with an error message from which we conclude that the feasible domain is empty. To further prevent a return to a known local solution, an RBF kernel function of the form

$$\rho \exp(-\mu \|x - x^{loc}\|^2)$$

is added to the objective function for each local solution.

However, an appropriate choice of the tolerances ϵ_1, ϵ_2, ρ, and μ depends on the distribution of the local solutions, scaling properties, and the curvature of the objective function. Moreover, the nonlinear programs generated this way, become more and more non-convex. Thus, one has to add an additional regularization to stabilize the algorithm and appropriate starting points for each subproblem.

In Section 2, we outline the heuristic procedure in more detail. Regularization and numerical aspects are discussed. Section 3 contains some numerical results obtained for a set of 40 standard test problems found in the literature. The usage of the Fortran subroutine is documented in Section 4 and Section 5 contains an illustrative example.

2 The Algorithm

We proceed from the constrained nonlinear program (1) without any information about the number of local minima. Note that the feasible set P is compact and the objective function $f(x)$ is continuous on P, i.e., we know that a global solution exists.

Let $x_1^\star$, ..., $x_k^\star$ be a series of known local minima, i.e., a series of feasible points satisfying the necessary KKT optimality conditions of (1). For computing a new iterate

$x^\star_{k+1}$, the following expanded nonlinear program is formulated and solved,

$$x \in \mathbb{R}^n : \quad \begin{array}{l} \min \; f(x) + \sum_{i=1}^{k} \rho_i \exp(-\mu_i \|x - x_i^\star\|^2) \\ g_j(x) = 0 \; , \quad j = 1, \ldots, m_e, \\ g_j(x) \geq 0 \; , \quad j = m_e + 1, \ldots, m, \\ f(x) \leq f_k^\star - \epsilon_1 |f_k^\star| \; , \\ \|x - x_i^\star\|^2 \geq \epsilon_2 \; , \quad i = 1, \ldots, k, \\ x_l \leq x \leq x_u \; . \end{array} \tag{2}$$

Here, $f_k^\star$ is the best known objective function value, i.e.,

$$f_k^\star := \min_{i=1,\ldots,k} f(x_i^\star) \quad , \tag{3}$$

and $\|.\|$ denotes the Euclidean norm. Once a local solution of (2) is found, the objective function value is cut away and a ball around the minimizer prevents an approximation of any of the previous iterates. In addition, a radial basis function (RBF) is added to the objective function to *push away* subsequent local minimizers from the known ones. $\epsilon_1 > 0$ and $\epsilon_2 > 0$ are suitable tolerances for defining the artificial constraints, also $\rho_i > 0$ and $\mu_i > 0$ for defining the RBF kernel.

If the applied solution method for the nonlinear program (2) terminates with an error message at a non-feasible iterate, we conclude that P is empty and stop. Otherwise, we obtain at least a feasible point $x^\star_{k+1}$ with an objective function value better than all known ones, and k is replaced by $k+1$ to solve (2) again.

Obviously, we get a series of feasible iterates with decreasing objective function values. However, the approach has a couple of severe drawbacks:

1. The choice of the tolerances ϵ_1 and ϵ_2 is critical for the performance and it is extremely difficult to find appropriate values in advance. Too small values prevent the algorithm from moving away from the neighborhood of a local minimizer of the original program (1) towards another local minimizer, and too large values could cut off too many local minimizers, even the global one.

2. The choice of the RBF kernel parameters ρ_i and μ_i seems to be less crucial, but must nevertheless carefully adapted to the structure on the model functions.

3. Even if the initial feasible set P is convex, the additional constraints are non-convex and the feasible domains of (2) become more and more irregular. It is even possible that an initial connected set P becomes non-connected.

4. The algorithm stops as soon as the constraints of (2) become inconsistent. But infeasibility cannot be checked by any mathematical criterion. The only possibility

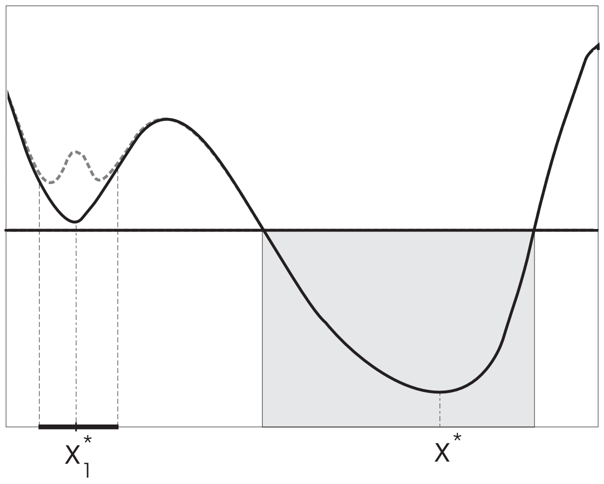

Figure 1: Local and Global Minimizer

is to run an optimization algorithm until an error message occurs at an infeasible iterate. But there is no guarantee in this case that the feasible region is in fact non-empty.

5. The local solutions of (2) do not coincide with local solutions of (1), if some of the artificial constraints become active.

6. It is difficult to find appropriate starting values for solving (2). Keeping the original one provided by the user, could force the iterates to get stuck at a previous local solution until an internal error occurs.

The situation is illustrated in Figure 2. Once a local solution $x_1^\star$ is obtained, an interval with radius ϵ_2 around $x_1^\star$ and a cut of the objective function subject to a relative bound ϵ_1 try to push away subsequent iterates from $x_1^\star$. The dotted line shows the objective function to be minimized, including the RBF term. The new feasible domain is shrinking, as shown by the gray area. If, however, the applied descent algorithm is started close to $x_1^\star$, it tries to follow the slope and to reach $x_1^\star$. If ϵ_1 and ϵ_2 are too small and if there are additional numerical instabilities, for example badly scaled functions or inaccurate gradient approximation, it is possible that the code runs into an error situation despite of its theoretical convergence properties.

To overcome at least some of these disadvantages, it is tried to regularize (2) in the following sense. For each of the artificial inequality constraints, a slack variable is introduced. Thus, we are sure that the feasible domain of the new subproblem is always non-empty. However, we get only a perturbed solution of (2) in case of non-zero slack variables. To reduce their size and influence as much as possible, a penalty term is added to the objective function, and we get the problem

$$x \in I\!R^n, y \in I\!R, z \in I\!R^k, : \begin{array}{l} \min \ \ f(x) + \sum_{i=1}^{k} \rho_i \exp(-\mu_i \|x - x_i^\star\|^2) + \gamma_k (y + e^T z) \\ g_j(x) = 0 \ , \ \ j = 1, \ldots, m_e, \\ g_j(x) \geq 0 \ , \ \ j = m_e + 1, \ldots, m, \\ f(x) \leq f_k^\star - \epsilon_1 |f_k^\star| + y \ , \\ \|x - x_i^\star\|^2 \geq \epsilon_2 - e_i^T z \ , \ \ i = 1, \ldots, k, \\ x_l \leq x \leq x_u \ , \\ 0 \leq y \leq \beta_1 \ , \\ 0 \leq z \leq \beta_2 \ . \end{array} \tag{4}$$

Here $e_i \in I\!R^k$ denotes the i-th unit vector, $i = 1, \ldots, k$, γ_k is a penalty parameter, and β_1 and β_2 are upper bounds for the slack variables y and z, $e = (1, \ldots, 1)^T$. By letting $\beta_1 = 0$ or $\beta_2 = 0$, the corresponding slack variables are suppressed completely.

There remains the question how to find a suitable starting point for (4) without forcing a user to supply too much information about the optimization problem and without trying to find some kind of pattern or decomposition of the feasible domain, where problem functions must be evaluated. Basically, the user should have full control how to proceed, and new starting values could be computed randomly. Another possibility is to choose

$$x_k^0 = \frac{1}{k+1} \left(x_0 + \sum_{i=1}^{k} x_i^\star \right) \tag{5}$$

where $x_0 \in I\!R^n$ is the initial starting point provided by the user.

The algorithm can be summarized as follows:

Algorithm 2.1 *Global Optimum of (1)*
<u>Start:</u> Select a starting point $x_0 \in I\!R^n$ and some tolerances $\rho > 0$, $\mu > 0$, $\epsilon_1 > 0$, $\epsilon_2 > 0$, $\epsilon_3 > 0$, $\beta_1 \geq 0$, $\beta_2 \geq 0$, $\gamma > 0$, and $\delta > 1$. Moreover, let k_{max} be an upper bound for the number of iterations.
<u>Initialization:</u> Solve (1) by a locally convergent optimization algorithm to get a local minimizer $x_1^\star$. Let $\gamma_1 := \gamma$, $\rho_1 = \rho$, and $\mu_1 = \mu$.
<u>Iteration Cycle:</u> For $k = 1, 2,, \ldots$ compute $x_{k+1}^\star$ from $x_1^\star$, $\ldots$, $x_k^\star$ as follows:

1. *Determine $f_k^\star$ by (3).*

2. *Formulate the expanded nonlinear program (4) with slack variables $y \in I\!R$ and $z \in I\!R^k$.*

3. *Solve (4) by an available locally convergent algorithm for smooth, constrained nonlinear programming starting from $x_k^0 \in I\!R^n$, for example given by (5), $y = 0$, and $z = 0$. Let x_{k+1}, y_{k+1}, and z_{k+1} be the optimal solution.*

4. *If $y_k + e^T z_k > \epsilon_3$, let $\gamma_{k+1} = \delta\gamma_k$, $\rho_{k+1} = \delta\rho_k$, and $\mu_{k+1} = \delta\mu_k$. Otherwise, let $\gamma_{k+1} = \gamma_k$, $\rho_{k+1} = \rho_k$, and $\mu_{k+1} = \mu_k$.*

5. *Let $k := k + 1$. If the maximum number of iterations is reached, i.e., if $k = k_{max}$, then stop.*

6. *Repeat the iteration, if the local optimization code reports that all internal convergence criteria are satisfied.*

7. *Stop otherwise. The last successful return is supposed to be the global minimizer.*

Another tolerance ϵ_3 is introduced to adopt the penalty parameter γ_k and to force the artificial variables y and z to become as small as possible. Usually we set $\epsilon_3 = \epsilon_1$.

The algorithm is a heuristic one without guaranteed convergence. However, there are many situations preventing the application of a more rigorous method, for example based on a partition technique, stochastic optimization, approximations, or a direct search method. The main disadvantage of these algorithms is the large number of function evaluations, often not acceptable for realistic time-consuming simulation programs. Especially nonlinear equality constraints are often not appropriate and must be handled in form of penalty terms, by which direct and random search algorithms can be deteriorated drastically. To summarize, the approach seems to be applicable under the subsequent conditions:

- The evaluation of the model functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$ is expensive.
- There are highly nonlinear restrictions, especially equality constraints.
- Model functions are continuously differentiable and the numerical noise in function and gradient evaluations is negligible.
- The number of local minima of (1) is not too large.
- There is some empirical, model-based knowledge about the expected relative locations of local minima and the curvature of the objective function.

3 Numerical Tests

The Fortran code for the heuristic global optimization methodology as outlined by Algorithm 2.1 is called NLPQLG. The nonlinear programming subprograms are solved by NLPQLP, version 2.2, an implementation of a sequential quadratic programming (SQP) algorithm, see Schittkowski [29]. The mathematical algorithm is described in Powell [21] and Schittkowski [23, 27] in more detail. NLPQLP is executed in reverse communication, where function and gradient values must be provided outside of the main subroutine depending on a flag. The only parameters that influence the performance of NLPQLP and that must be set by the user, are the maximum number of iterations (MAXIT) and the termination accuracy (ACC). In our case, we use MAXIT=500 and ACC=10^{-6} for all test runs. The remaining parameters for calling NLPQLG, see also the general model structure (4), are

first RBF kernel parameter (height):	ρ	$=$	100
second RBF kernel parameter (deviation):	μ	$=$	10^6
bound for cutting functions values:	ϵ_1	$=$	0.1
bound for cutting known minimizers:	ϵ_2	$=$	0.0001
percentage of cutting local minima:	β_1	$=$	10^{10}
upper bound for slack variable x:	β_2	$=$	0
penalty term for additional variables:	γ	$=$	10
factor for increasing penalty term:	δ	$=$	10

Here we use a small tolerance ϵ_2 for staying away from known local minimizers without regularization. The numerical tests are based on a collection of 58 test problems, all of them listed in Section 3 in full detail. A summary is presented in Table 3, where the following data are displayed:

no	-	test problem number,
name	-	name of the test problem,
reference	-	citation of literature,
n	-	number of variables,
m	-	number of constraints,
m_e	-	number of equality constraints,
n_{loc}	-	number of known local solutions,
$f^\star$	-	best known objective function value,

If $n_{loc} = 0$, the number of local solutions is not known. The first eight examples are from two collections of nonlinear programming test problems for gradient-based locally convergent methods, see Hock and Schittkowski [13] and Schittkowski [25]. But in these cases, the applied nonlinear programming code NLPQLP stopped at a local solution, and we are interested in the question, whether a global minimizer can be obtained or not. The numerical results of NLPQLP for all 306 test problems are published in Schittkowski [28].

Usually, global optimization algorithms do not depend on a single starting point as is the case for local methods. Thus, most test problems found in the literature do not contain any information from where a successive local search code could be started. Section 3 contains the test problems together with our more or less arbitrarily chosen initial guesses. To avoid approximation of the global minimizer in the first step by solving the non-perturbed problem, they are chosen sufficiently far away from the global optimum. Subsequently, new starting points are selected randomly between the given upper and lower bounds.

Since the Branin-problem has four local minimizers with same function values, the initial search step leads to a global solution. Only in case of problems Pinter 1 and

no	name	reference	n	m	m_e	n_{loc}	$f^\star$
1	TP16	Hock, Schittkowski [13]	2	2	0	0	0.250
2	TP33	Hock, Schittkowski [13]	3	2	0	0	-4.586
3	TP54	Hock, Schittkowski [13]	6	1	1	0	-0.908
4	TP55	Hock, Schittkowski [13]	6	6	6	0	6.333
5	TP59	Hock, Schittkowski [13]	2	3	0	0	-7.804
6	TP202	Schittkowski [25]	2	0	0	0	1.000
7	TP220	Schittkowski [25]	2	1	1	0	1.000
8	TP265	Schittkowski [25]	4	2	2	0	0.975
9	Rastrigin	Rastrigin [22], Törn, Zilinskas [32]	2	0	0	50	-2.000
10	Adjiman	Adjiman et al. [1], Floudas et al. [5]	2	0	0	3	-2.022
11	Six-Hump Camel	Branin [2], Törn, Zilinskas [32]	2	0	0	6	-1.032
12	Murtagh, Saunders	Murtagh, Saunders [18], Floudas et al. [5]	5	3	3	5	0.029
13	Branin	Branin, Hoo [3], Törn, Zilinskas [32]	2	0	0	3	0.398
14	Goldstein, Price	Goldstein, Price [7], Floudas et al. [5]	2	0	0	4	3.000
15	Shekel 2	Dixon, Szegö [4], Törn, Zilinskas [32]	4	0	0	0	-
16	Shekel 5	Dixon, Szegö [4], Törn, Zilinskas [32]	4	0	0	0	-
17	Shekel 7	Dixon, Szegö [4], Törn, Zilinskas [32]	4	0	0	0	-
18	Shekel 10	Dixon, Szegö [4], Törn, Zilinskas [32]	4	0	0	0	-
19	Griewank 2	Griewank [9], Törn, Zilinskas [32]	2	0	0	> 500	1.000
20	Griewank 10	Griewank [9], Törn, Zilinskas [32]	10	0	0	> 1000	1.000
21	Pinter 1	Pinter [19]	50	0	0	0	1.000
22	Pinter 2	Pinter [19]	50	0	0	0	1.000
23	Pinter 3	Pinter [20]	5	4	2	0	1.000
24	Sinus		2	1	1	0	-10.000
25	Hesse	Hesse [11], Törn, Zilinskas [32]	6	6	0	18	-310.000
26	Polynomial	Wang, Dong, Aitchison [34]	2	0	0	≥ 5	0.000
27	Hartman 3	Hartman [10], Törn, Zilinskas [32]	3	0	0	≥ 3	-3.860
28	Hartman 6	Hartman [10], Törn, Zilinskas [32]	6	0	0	≥ 3	-3.320
29	Equations	Pinter [19]	3	0	0	0	1.000
30	Strongin	Strongin [31], Törn, Zilinskas [32]	2	0	0	0	1.000
31	$x \sin(x)$		2	0	0	0	-18.000
32	Trefethen	Trefethen [33]	2	0	0	>1000	-3.307
33	Gomez 2	Gomez, Levy [8]	2	1	0	0	1.000
34	Gomez 3	Gomez, Levy [8]	2	1	0	0	-0.971
35	Floudas, Pardalos 9	Floudas et al. [5]	2	2	0	0	-5.508
36	Ackley Path 2	Storn, Price [30]	2	0	0	0	1.0
37	Easom	Michalewicz [17]	2	0	0	0	-1.0
38	2D Sinus		2	0	0	0	1.0
39	Zabinsky	Zabinsky et al. [35]	10	0	0	>1000	-3.5
40	Ackley Path 10	Storn, Price [30]	10	0	0	0	1.0
41	Pinter constrained 1	Pinter [20]	2	3	0	0	1.0
42	Pinter constrained 2	Pinter [20]	2	4	0	0	1.0
43	Pinter constrained 3	Pinter [20]	2	0	0	0	1.0
44	Pinter constrained 4	Pinter [20]	2	0	0	0	1.0
45	Pinter constrained 5	Pinter [20]	2	0	0	0	1.0
46	Largest small polygon		40	211	2	0	1.0
47	Electrons in sphere		150	50	50	0	1.0
48	Haverly Pooling problem		9	6	4	0	-600.0
49	Rastrigin 20	Rastrigin [22], Törn, Zilinskas [32]	20	0	0	0	1.0
50	Rastrigin 50	Rastrigin [22], Törn, Zilinskas [32]	50	0	0	0	1.0
51	scaled Rastrigin 20	Rastrigin [22], Törn, Zilinskas [32]	20	0	0	0	1.0
52	scaled Rastrigin 50	Rastrigin [22], Törn, Zilinskas [32]	50	0	0	0	1.0
53	Levy 20	Gomez, Levy [8]	20	0	0	0	1.0
54	Levy 50	Gomez, Levy [8]	50	0	0	0	1.0
55	Ackley Path 20	Storn, Price [30]	20	0	0	0	-23.
56	Ackley Path 50	Storn, Price [30]	50	0	0	0	-23.
57	Schwefel 5		5	0	0	0	-418.983 n
58	Schwefel 10		10	0	0	0	-418.983 n

Table 1: Test Problems: Dimensioning Parameters and Tolerances

Pinter 2, we were unable to find initial values not leading the local search directly to the global minimizer. In case of zero global objective function value, we added the constant 1.0 to $f(x)$ to avoid irregularities, see Algorithm 2.1.

Since analytical derivatives are not available for all problems, we approximate them numerically by a fourth-order difference formula,

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{4!\eta_i}\Big(2f(x-2\eta_i e_i) - 16f(x-\eta_i e_i) + 16f(x+\eta_i e_i) - 2f(x+2\eta_i e_i)\Big) \qquad (6)$$

where $\eta_i = \eta \max(\eta, |x_i|)$, $\eta = 10^{-6}$, e_i the i-th unit vector, and $i = 1, \ldots, n$. In the same way, derivatives of constraints are computed.

For most test examples, the exact global solution is known in advance. For the Shekel-problems, only approximations of the global solution, $x_i = 1/c_i$, $i = 1, \ldots, n$, are known. Global solutions for the first eight problems from the standard test problem collections are the best known local solutions, see Schittkowski [28] for details.

The numerical results are summarized in Table 3. We report the test problem number, the index j of the local search cycle where the best feasible solution is found, the total number of local searches, n_{it}, i.e., of calls of NLPQLP, and the total number of function and gradient evaluations, n_f and n_g, respectively. To give an impression on the improvements from starting point over the first local solution, also the objective function values $f(x_0)$, $f(x_1^\star)$, $f(x_j^\star)$, and $f(x^\star)$ are listed, where x_0 is the starting point, $x_1^\star$ the first local solution, $x_j^\star$ the best feasible solution obtained, and $x^\star$ the known or guessed global solution, respectively. Note that the starting point x_0 is infeasible in some situations.

In most cases, the global minimal value is approximated subject to an acceptable error ϵ_g as shown in the second last column of Table 3. However, flat objective function areas or a large number of local solutions can prevent the computation of the exact global minimizer by our approach, at least when proceeding from the parameters given. Thus, we present also the percentage by which a global solution is approximated relative to the first local minimizer found, called p_r.

In case of problems 4, 22, and 23, the algorithm stopped at the first local minimizer found. Further improvements are not possible, probably because of a too big ϵ_1. For example 7, an improvement of the first local minimizer found is not possible. The problem is irregular, i.e., the constraint qualification is not satisfied at the optimal solution. For problems 9, 19, 20, and 34, the known minimum value is not found, but we observe a significant improvement of the first local minimizer. In case of problem 32, there are more than 667 local minima only in the subarea between -1 and 1. The first local minimizer is improved by more than 50 percent. For problems 16, 17, and 18, the exact local minimal values are not known. It is not possible to solve problem 31.

The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 9.1, under Windows XP, and executed on a Pentium IV processor with 2.8 GHz..

no	j	n_{it}	n_f	n_g	$f(x_0)$	$f(x_1^\star)$	$f(x_j^\star)$	$f(x^\star)$	ϵ_g	p_r	
1	2	14	334	299	2	910.	3.9820619	0.25000000	0.25000000	0.00	0.00
2	2	6	168	158	2	2.59	-4.0000000	-4.5857864	-4.5857864	0.581E-14	0.00
3	5	10	108	87	2	1.14	-0.89000352E-33	-0.90807475	-0.90807476	0.573E-08	0.00
4	1	0	51	1	4	0.667	0.66666651	6.0000000	6.3333335	-0.526E-01	0.00
5	2	5	1560	560	2	73.7	-6.7495052	-7.8027921	-7.8042263	0.184E-03	0.00
6	4	5	218	142	2	0.126E+04	49.984254	1.0000006	1.0000000	0.564E-06	0.00
7	4	6	3338	564	1	839.	838.96580	-837.96577	-837.96580	0.304E-07	0.00
8	2	5	88	75	2	0.253E-01	1.9036248	0.97474660	0.97474658	0.188E-07	0.00
9	11	12	316	104	2	4.11	0.67936658	-2.0000000	-2.0000000	0.230E-07	0.00
10	3	9	222	199	2	3.07	-0.99494502	-2.0218067	-2.0218101	0.165E-05	0.00
11	5	7	199	155	2	0.642E+04	0.19851038E-07	-1.0316285	-1.0316300	0.154E-05	0.00
12	3	5	157	132	2	37.0	27.871905	0.29318316E-01	0.29310831E-01	0.255E-03	0.00
13	4	9	267	200	2	56.2	0.39788769	0.39788736	0.39788753	-0.428E-06	0.00
14	2	8	272	204	2	0.767E+05	84.000147	3.0000791	3.0000000	0.264E-04	0.00
15	2	3	803	485	2	11.1	11.075976	-10.027624	-10.086000	0.579E-02	0.00
16	2	6	507	318	2	10.8	-5.0551977	-10.153200	-10.086000	-0.666E-02	0.00
17	4	5	355	235	2	11.0	-2.7519322	-10.402940	-10.086000	-0.314E-01	0.00
18	7	7	444	258	2	11.0	-2.8702184	-10.522312	-10.086000	-0.433E-01	0.00
19	13	13	281	193	2	102.	13.140931	3.4514297	1.0000000	2.45	20.19
20	3	10	1435	570	2	27.0	2.4323896	1.0003478	1.0000000	0.348E-03	0.00
21	15	19	4855	2209	2	0.704E+06	1.0006594	1.0000000	1.0000000	0.421E-08	0.00
22	3	4	2522	1139	2	0.144E+06	1.0001375	1.0000066	1.0000000	0.661E-05	0.00
23	1	0	89	58	2	18.5	1.0000000	1.0000000	1.0000000	0.358E-07	0.00
24	2	4	43	30	2	10.8	10.833333	-9.9999999	-9.9999999	0.882E-12	0.00
25	1	10	132	130	2	67.0	-298.00000	-298.00000	-310.00000	0.387E-01	100.00
26	2	6	110	85	2	0.101E+09	15.203125	1.0000006	1.0000000	0.637E-06	0.00
27	2	3	83	56	2	3.58	-0.17021103E-04	-4.2188387	-4.2189999	0.382E-04	0.00
28	2	5	383	246	2	4.33	-3.2323045	-3.3342357	-3.3340001	-0.707E-04	0.00
29	7	11	554	380	2	21.4	5.6402016	1.0000002	1.0000000	0.212E-06	0.00
30	8	8	762	225	2	52.0	51.960769	1.0000000	1.0000000	0.126E-10	0.00
31	6	6	89	59	2	23.2	-4.0866490	-9.3716440	-18.000000	0.479	62.01
32	9	13	448	170	2	5.00	-0.53883390	-3.1440794	-3.3068690	0.492E-01	5.88
33	10	10	276	200	2	1.05	1.0465290	1.0000000	1.0000000	0.467E-10	0.00
34	2	7	149	123	2	2.22	0.21749159	-0.90840131	-0.97109997	0.646E-01	5.28
35	3	5	100	86	2	5.71	-4.0537078	-5.5080133	-5.5079002	-0.205E-04	0.00
36	6	11	360	153	2	20.8	20.839778	18.553659	1.0000000	17.6	88.48
37	9	17	38	36	2	2.00	2.0000000	-0.99999982	-1.0000000	0.183E-06	0.00
38	6	6	169	116	2	120.	5.1353480	1.0000026	1.0000000	0.262E-05	0.00
39	6	8	494	249	2	3.50	3.4999801	-3.4999998	-3.5000000	0.446E-07	0.00
40	11	14	1075	408	2	8.31	8.3052609	3.4119178	1.0000000	2.41	33.02
41	2	2	27	23	2	1.05	4.4551122	1.0000000	1.0000000	0.112E-07	0.00
42	1	1	14	12	2	1.05	4.4551122	4.4551122	1.0000000	3.46	100.00
43	6	9	378	263	2	3.57	1.0000001	1.0000000	1.0000000	0.344E-11	0.00
44	2	9	469	254	2	8.73	4.6394326	1.0000000	1.0000000	0.206E-09	0.00
45	11	12	1828	841	2	60.3	17.990157	1.0000006	1.0000000	0.565E-06	0.00
46	12	12	894	599	2	0.959	0.26183739	0.22937506	1.0000000	-0.771	0.00
47	8	8	560	363	2	1.00	1.0015563	1.0013678	1.0000000	0.137E-02	0.00
48	2	2	59	57	2	619.	-400.00000	-600.00000	-600.00000	0.00	0.00
49	6	9	2033	612	2	217.	19.904224	13.934468	1.0000000	12.9	68.42
50	4	11	3670	1109	2	483.	49.752995	40.798369	1.0000000	39.8	81.63
51	3	10	2563	775	2	250.	19.904224	13.934470	1.0000000	12.9	68.42
52	1	7	2605	839	2	484.	54.727760	54.727760	1.0000000	53.7	100.00
53	4	6	377	265	2	116.	8.9495157	1.0000055	1.0000000	0.547E-05	0.00
54	5	6	567	408	2	298.	8.9495438	1.0000000	1.0000000	0.243E-08	0.00
55	6	11	1474	462	2	20.3	18.964180	18.459431	1.0000000	17.5	97.19
56	3	11	2680	742	2	20.7	18.936733	18.551288	1.0000000	17.6	97.85
57	5	13	500	346	2	1.25	-0.12770309	-0.16408975	-0.20949146	0.217	55.51
58	4	11	473	284	2	1.45	1.4488996	-0.29609392	-0.41898291	0.293	6.58

Table 2: Numerical Results

To summarize, we evaluate the percentage of successful test runs p_{succ} subject to a relative tolerance of 0.01 by which the known global solution is approximated, the percentage of test runs where the global solution is at least approximated within 5 % compared to the first local solution, p_{imp}, the average number of subproblems (4) solved until the global minimizer is found, n_{glob}, the average number of subproblems solved, n_{tot}, the average number of function calls, n_f, and the average number of gradients calls, n_g,

$$\begin{array}{lcrl} p_{succ} & = & 72.5 & \% \\ p_{imp} & = & 94.8 & \% \\ n_{glob} & = & 4.8 & \\ n_{tot} & = & 8.0 & \\ n_f & = & 776 & \\ n_g & = & 333 & \end{array}$$

4 Program Documentation

NLPQLG is implemented in form of a Fortran subroutine, where the nonlinear programming subproblem is solved by NLPQLP, see Schittkowski [29]. Model functions and gradients are provided by reverse communication according to the following rules:

1. Choose starting values for the variables to be found, and store them in X.
2. Compute objective and all constraint function values, store them in F and 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 NLPQLG.
5. If NLPQLG returns with IFAIL=-1, compute objective function and constraint values for variable values found in X, store them in F and G, and call NLPQLG again.
6. If NLPQLG terminates with IFAIL=-2, compute gradient values with respect to the variables stored in X, and store them in DF and DG. Then call NLPQLG again.
7. If NLPQLG terminates with IFAIL=-3, compute new starting values and corresponding objective function values, constraint function values, and all gradients of these function, and store them in F, G, DF, and DG, respectively. Then call NLPQLG again.
8. If NLPQLG terminates with IFAIL>0, we are done. An improved local minimized is hopefully found in a previous iteration.

Note that the iteration cycle of Algorithm 2.1 is only stopped if either the maximum number of outer iterations, LMAX, is reached, or if the internal execution of NLPQLP is interrupted caused by an error situation. In this case, it is supposed that the feasible domain became inconsistent. Thus, NLPQLG will always stop with an error message, usually the termination reason of the last execution of NLPQLP.

If NLPQLG is called with LMAX=1, the execution is exactly the same as when calling the nonlinear programming routine NLPQLP only once.

Usage:

```
      CALL   NLPQLG (        M,      ME,     MMAX,      N,    NMAX,
  /                          L,    LMAX,   MNNMAX,     X0,       X,
  /                         XS,       F,    FINIT,     FS,       G,
  /                         GS,      JS,       DF,     DG,       U,
  /                         XL,      XU,        C,      D,     ACC,
  /                      ACCQP,    EPS1,     EPS2,   BND1,    BND2,
  /                      ALAMB,     RHO,     AMUE,  ALFAC,   SCBOU,
  /                     MAXFUN,   MAXIT,   IPRINT,   IOUT,   IFAIL,
  /                         WA,     LWA,      KWA,   LKWA,  ACTIVE,
  /                     LACTIV                                     )
```

Definition of the parameters:

M :	Number of constraints.
ME :	Number of equality constraints.
MMAX :	Row dimension of array DG containing Jacobian of constraints. MMAX must be at least one and greater or equal to M+LMAX+1.
N :	Number of optimization variables.
NMAX :	Row dimension of C. NMAX must be at least two and greater than N+LMAX+1.
L :	On return, L shows the number of local minima found.
LMAX :	Maximum number of expected local minima. LMAX must be at least 1. In this case, the first local minimizer is searched.

MNNMAX :	Dimensioning parameter, must be at least MMAX+NMAX+NMAX.
X0(N) :	When calling NLPQLG, X0 has to contain starting values for the first call of the SQP algorithm.
X(NMAX) :	On return, X contains the best local solution obtained. In the driving program, the row dimension of X must be NMAX.
XS(NMAX,LMAX) :	On return, XS contains L local minimizers found. In the driving program the row dimension of XS must be equal to NMAX.
F :	On return, F contains the objective function value of the best local minimizer including penalty term for artificial variables.
FINIT :	On return, FINIT stores the first local minimizer of the unperturbed problem.
FS(LMAX) :	On return, FS contains L objective function values of the local minima stored in XS.
G(MMAX) :	On return, G contains the constraint function values at the best local solution expanded by artificial constraints. In the driving program, the dimension of G must be equal to MMAX.
GS(MMAX,LMAX) :	On return, GS contains the constraint values of all L local minizers found. In the driving program, the row dimension of GS must be equal to MMAX.
JS :	On return, JS contains the index of the best solution in XS, FS and GS, respectively.
DF(NMAX) :	DF is a working array for the current gradient of the objective function of the expanded problem.
DG(MMAX,NMAX) :	DG is a working array for gradients of constraints. In the driving program the row dimension of DG has to be equal to MMAX.
U(MNNMAX) :	U is a working array for multipliers. The first M locations contain the multipliers of the M nonlinear constraints.
XL(NMAX),	On input, the one-dimensional arrays XL and XU must contain the
XU(NMAX) :	upper and lower bounds of the variables.
C(NMAX,NMAX) :	C is a working array for the BFGS approximation of the Hessian matrix of the Lagrangian function stored in form of an LDL decomposition. C contains the lower triangular factor of an LDL factorization of the final quasi-Newton matrix (without diagonal elements, which are always one). In the driving program, the row dimension of C has to be equal to NMAX.

D(NMAX) :	The elements of the diagonal matrix of the LDL decomposition of the quasi-Newton matrix are stored in the one-dimensional array D.
ACC :	The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be 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 internally and subsequently multiplied by 10.0.
EPS1 :	Relative improvement of local minimizer subject to the best value found in the previous iterates, needed for definition of one additional constraint. EPS1 must be greater than zero (e.g. 0.01).
EPS2 :	Absolute distance bound for preventing approximation of a known local minimizer. EPS2 must be greater than zero (e.g. 0.1).
BND1 :	Upper bound for artificial variable X(N+1) for relative improvement of local minimizer subject to the best value known. BND1 must not be smaller than zero.
BND2 :	Upper bound for artificial variables X(N+1+I), I=1,...,K-1, for distances from known local minimizers. BND2 must not be smaller than zero.
ALAMB :	Common penalty parameter to keep the additionally introduced variables X(N+1),...,X(N+K) as small as possible. ALAMB must be positive (e.g. 100.0).
RHO :	RBF parameter, height or penalty coefficient, respectively.
AMUE :	RBF parameter, standard distribution.
ALFAC :	Factor greater than one to increase ALAMB in case of any positive artificial variables (e.g. 10.0).
SCBOU :	Allows automatic scaling of all problem functions at the starting point X0. If a function value is greater than SCBOU, function values are divided by their square root (e.g. 1.0).

MAXFUN :	The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20).
MAXIT :	Maximum number of iterations for each subproblem, where one iteration corresponds to one formulation and solution of a quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g. 100).
IPRINT :	Specification of the desired output level. 0 - no output of the program 1 - only a final summary of local solutions 2 - final convergence analysis for each subproblem 3 - one line of intermediate results per iteration 4 - more detailed information per iteration
IOUT :	Integer indicating the desired output unit number, i.e., all write-statements start with 'WRITE(IOUT,... ' in case of IPRINT>0.
IFAIL :	The parameter shows the reason for terminating the last iteration cycle. Initially, IFAIL must be set to zero. On return, IFAIL indicates the termination reason, greater than zero in case of stopping by an error, or less than zero to perform reverse communication as outlined above. In case of IFAIL=0, an optimal solution of one of the auxiliary problems was obtained. The following error messages are possible: < 0 - reverse communication, see above 0 - successful termination 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, M>MMAX, N$\geq$NMAX, or MNN2$\neq$M+N+N+2 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
	>100 - error message of QP solver
WA(LWA) :	WA is a real working array of length LWA.
LWA:	Length of WA, should be at least 3*NMAX*NMAX/2 + 11*MMAX + 33*NMAX + 2*LMAX + 200.
KWA(LKWA):	The user has to provide working space for an integer array of length LKWA.
LKWA :	Length of KWA, should be at least NMAX + 30.
ACTIVE(LACTIV) :	The logical array shows a user the constraints, which NLPQLP considers to be active at the last computed iterate, i.e. G(J,X) is active, if and only if ACTIVE(J)=.TRUE., J=1,...,M.
LACTIV :	Length of ACTIVE, should be at least 2*MMAX + 10.

5 An Illustrative Example

To give an example how to organize the code, we consider the Murtagh, Saunders [18] problem,

$$x_1, ..., x_5 \in I\!R : \begin{array}{l} \min\ (x_1-1)^2 + (x_1-x_2)^2 + (x_2-x_3)^3 + (x_3-x_4)^4 + (x_4-x_5)^4 \\ x_1 + x_2^2 + x_3^3 - 3\sqrt{2} - 2 = 0\ , \\ x_2 - x_3^2 + x_4 - 2\sqrt{2} + 2 = 0\ , \\ -5 \le x_i \le 5, i = 1, ..., 5 \end{array} \tag{7}$$

The Fortran source code for executing NLPQLG is listed below. New starting values are selected randomly.

```
      IMPLICIT          NONE
      INTEGER           NMX, MMX, LMAX, NMAX, MMAX, MNNMAX, LWA, LKWA,
     /                  LACTIV
      PARAMETER        (NMX = 6, MMX = 3, LMAX = 5)
      PARAMETER        (NMAX   = NMX + LMAX + 1,
     /                  MMAX   = MMX + LMAX + 1,
     /                  MNNMAX = MMAX + NMAX + NMAX + 2,
     /                  LWA    = 3*NMAX*NMAX/2 + 33*NMAX + 11*MMAX
     /                           + 2*LMAX + 200,
     /                  LKWA   = NMAX + 30,
     /                  LACTIV = 2*MMAX + 10)
      DOUBLE PRECISION  X(NMAX),X0(NMAX),XS(NMAX,LMAX),G(MMAX),DF(NMAX),
     /                  DG(MMAX,NMAX),U(MNNMAX),FS(LMAX),GS(MMAX,LMAX),
     /                  XL(NMAX),XU(NMAX),C(NMAX,NMAX),D(NMAX),
     /                  WA(LWA),KWA(LKWA), ACC, ACCQP, EPS1, EPS2,
     /                  ALFAC, BND1, BND2, ALAMB, RHO, AMUE, SCBOU,
     /                  F, FINIT, RN
      INTEGER           M, ME, N, I, IOUT, MAXIT, MAXFUN, IPRINT,
     /                  IFAIL, L, JS
      LOGICAL           ACTIVE(LACTIV)
 C
C   Problem parameters:
C
      N  = 5
      M  = 3
      ME = 3
      DO I = 1,N
         X0(I) = -5.0D0
         X(I)  = X0(I)
         XL(I) = -5.0D0
         XU(I) = 5.0D0
      ENDDO
C
C   Tolerances for calling NLPQLG
C
      IOUT   = 6
      ACC    = 1.0D-10
      ACCQP  = 1.0D-14
      EPS1   = 0.1D0
      EPS2   = 0.1D0
      ALFAC  = 1.0D+1
      BND1   = 1.0D+10
      BND2   = 1.0D+10
      ALAMB  = 1.0D+1
      SCBOU  = 1.0D+0
      RHO    = 1.0D-6
      AMUE   = 1.0D+3
      MAXIT  = 500
      MAXFUN = 25
      IPRINT = 3
C
C   Execute NLPQLG in reverse communication
C
      IFAIL = 0
   10 CONTINUE
C
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-1).OR.(IFAIL.EQ.-3)) THEN
         F = (X(1)-1.0D0)**2 + (X(1)-X(2))**2 + (X(2)-X(3))**3
     /          + (X(3)-X(4))**4 + (X(4)-X(5))**4
```

```
         G(1) = X(1) + X(2)**2 + X(3)**3 - 3.0D0*DSQRT(2.0D0) - 2.0D0
         G(2) = X(2) - X(3)**2 + X(4) - 2.0D0*DSQRT(2.0D0) + 2.0D0
         G(3) = X(1)*X(5) - 2.0D0
      ENDIF
C
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-2).OR.(IFAIL.EQ.-3)) THEN
         DF(1) = 2.0D0*(X(1)-1.0D0) + 2.0D0*(X(1)-X(2))
         DF(2) = -2.0D0*(X(1)-X(2)) + 3.0D0*(X(2)-X(3))**2
         DF(3) = -3.0D0*(X(2)-X(3))**2 + 4.0D0*(X(3)-X(4))**3
         DF(4) = -4.0D0*(X(3)-X(4))**3 + 4.0D0*(X(4)-X(5))**3
         DF(5) = -4.0D0*(X(4)-X(5))**3
         DG(1,1) = 1.0D0
         DG(1,2) = 2.0D0*X(2)
         DG(1,3) = 3.0D0*X(3)**2
         DG(1,4) = 0.0D0
         DG(1,5) = 0.0D0
         DG(2,1) = 0.0D0
         DG(2,2) = 1.0D0
         DG(2,3) = -2.0D0*X(3)
         DG(2,4) = 1.0D0
         DG(2,5) = 0.0D0
         DG(3,1) = X(5)
         DG(3,2) = 0.0D0
         DG(3,3) = 0.0D0
         DG(3,4) = 0.0D0
         DG(3,5) = X(1)
      ENDIF
C
      CALL NLPQLG (      M,     ME,   MMAX,      N,   NMAX,
     /                   L,   LMAX, MNNMAX,     X0,      X,
     /                  XS,      F,  FINIT,     FS,      G,
     /                  GS,     JS,     DF,     DG,      U,
     /                  XL,     XU,      C,      D,    ACC,
     /               ACCQP,   EPS1,   EPS2,   BND1,   BND2,
     /               ALAMB,    RHO,   AMUE,  ALFAC,  SCBOU,
     /              MAXFUN,  MAXIT, IPRINT,   IOUT,  IFAIL,
     /                  WA,    LWA,    KWA,   LKWA, ACTIVE,
     /              LACTIV)
C
      IF (IFAIL.EQ.-3) THEN
         DO I = 1,N
            CALL RANDOM_NUMBER(RN)
            X(I) = XL(I) + RN*(XU(I) - XL(I))
         ENDDO
      ENDIF
      IF (IFAIL.LT.0) GOTO 10
C
      STOP 'DEMO_G'
      END
```

After five calls of NLPQLP, the global minimizer is reached. The following output should appear on screen:

```
    --------------------------------------------------------------------
     START OF THE GLOBALIZED SQP ALGORITHM
    --------------------------------------------------------------------

     Parameters:
        ACC    =     0.10D-09
        ACCQP  =     0.10D-13
        EPS1   =     0.10D+00
        EPS2   =     0.10D+00
        BND1   =     0.10D+11
        BND2   =     0.10D+11
        ALAMB  =     0.10D+02
        RHO    =     0.10D-05
        AMUE   =     0.10D+04
        ALFAC  =     0.10D+02
        SCBOU  =     0.10D+01
        MAXFUN =   25
        MAXIT  =  500
        IPRINT =    1

     --- Summary of Local Solutions ---

     Local solution ...    1
        Function value:  F(X) =  0.44022072D+02
        Variable:        X    =
             -0.70339280D+00  0.26357026D+01 -0.96361667D-01 -0.17979899D+01
             -0.28433615D+01

     Local solution ...    2
        Function value:  F(X) =  0.60703552D+03
        Variable:        X    =
             -0.27908708D+01 -0.30041386D+01  0.20537568D+00  0.38747449D+01
             -0.71662222D+00  0.00000000D+00  0.00000000D+00
```

```
Local solution ...    3
   Function value:  F(X) =  0.29310831D-01
   Variable:        X    =
         0.11166347D+01  0.12204407D+01  0.15377854D+01  0.19727704D+01
         0.17910961D+01  0.00000000D+00  0.00000000D+00  0.00000000D+00

Local solution ...    4
   Function value:  F(X) =  0.52902580D+02
   Variable:        X    =
         0.72800348D+00 -0.22452109D+01  0.77951383D+00  0.36812798D+01
         0.27472396D+01  0.00000000D+00  0.00000000D+00  0.00000000D+00
         0.70907396D-35

Local solution ...    5
   Function value:  F(X) =  0.44246295D+02
   Variable:        X    =
        -0.71101432D+00  0.26383655D+01 -0.19414179D+00 -0.17722473D+01
        -0.28128829D+01  0.00000000D+00  0.00000000D+00  0.00000000D+00
         0.00000000D+00  0.00000000D+00

Best solution at J = 3 out of L = 5 local solutions:

   Termination reason:           IFAIL =     0
   Number of function calls:     NFUNC =   115
   Number of gradient calls:     NGRAD =   100
   First local solution:         F(X1) =   0.44022072D+02
   Best function value:          F(X)  =   0.29310831D-01
   Sum of artificial variables:  FEAS  =   0.00000000D+00
   Sum of constraint violations: GVIOL =   0.21337847D-08
   Final penalty parameter:      ALAMB =   0.10000000D+02
   RBF parameter, height:        RHO   =   0.10000000D-05
   RBF parameter, deviation:     AMUE  =   0.10000000D+04

   Variable values:              X     =
         0.11166347D+01  0.12204407D+01  0.15377854D+01  0.19727704D+01
         0.17910961D+01

   Constraint values:            G(X)  =
         0.18692958D-09 -0.37589043D-10 -0.19092661D-08
```

6 Summary

We present a heuristic approach for stepwise approximation of the global solution of a constrained nonlinear programming problem. In each step, additional variables and constraints are added to the original ones, to cut off known local solutions. The idea is implemented and the resulting code NLPQLG is able to solve a large number of test problems found in the literature. However, the algorithm is quite sensitive subject to the input tolerances, which must be chosen very carefully. But under certain circumstances, for example very time-consuming function evaluations or highly nonlinear constraints, the proposed idea is often the only way to improve a known local solution.

References

[1] Adjiman C.S., Dallwig S., Floudas C.A., Neumaier A. (1998): *A global optimization method, αBB, for general twice-differentiable NLPs - I. Theoretical advances*, Computational Chemical Engineering, Vol. 9, 23-40

[2] Branin F.H. (1972): *Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations*, IBM Journal of Research Developments, 504-522

[3] Branin F.H., Hoo S.K. (1972): *A method for finding multiple extrema of a function of n variables*, in: F.A. Lootsma (ed.), Numerical Methods of Nonlinear Optimization, Academic Press, London, 231-237

[4] Dixon L.C.W., Szegö G.P. (1978): *The global optimization problem: an introduction*, in: Dixon L.C.W., Szegö G.P. (eds.), Towards Global Optimization 2, North-Holland, Amsterdam, 1-15

[5] Floudas C.A., Pardalos P.M., Adjiman C.S., Esposito W.R., Gümüs Z.H., Harding S.T., Klepeis J.L., Meyer A.A., Schweiger C.A. (1999): *Handbook of Test Problems in Local and Global Optimization*, Kluwer Academic Publishers, Dordrecht

[6] Ge R.P., Qin Y.F. (1987): *A class of filled functions for finding global minimizers of a function of several variables*, Journal of Optimization Theory and Applications, Vol. 54, 241-252

[7] Goldstein A., Price J. (1971): *On descent from local minima*, Mathematics of Computation, Vol. 25, 569-574

[8] Gomez S., Levy A. (1982): *The tunnelling method for solving the constraint global optimization problem with several non-connected feasible domains*, in: Dold A., Eckmann B. (eds.): Lecture Notes in Mathematics, Vol. 909, Springer, 34-47

[9] Griewank A. (1981): *Generalized descent for global optimization*, Journal of Optimization Theory and Applications, Vol. 34, 11-39

[10] Hartman J.K. (1973): *Some experiments in global optimization*, Naval Research Logistics Quaterly, Vol. 20, 569-576

[11] Hesse R. (1973): *A heuristic search procedure for estimating a global solution of nonconvex programming problems*, Operations Research, Vol. 21, 1267-1280

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

[13] Hock W., Schittkowski K. (1983): *A comparative performance evaluation of 27 nonlinear programming codes*, Computing, Vol. 30, 335-358

[14] Horst R., Pardalos P.M. eds. (1995): *Handbook of Global Optimization*, Kluwer Academic Publishers, Dordrecht

[15] Hough P.D., Kolda T.G., Torczon V.J. (2001): *Asynchronous parallel pattern search for nonlinear optimization*, to appear: SIAM Journal on Scientific Computing

[16] Levy A.V., Montalvo A. (1985): *The tunneling algorithm for the global minimization of functions*, SIAM Journal on Scientific and Statistical Computing, Vol. 6, 15-29

[17] Michalewicz Z. (1996): *Genetic Algorithms + Data Structures = Evolution Programming*, Springer, Berlin, Heidelberg, New York

[18] Murtagh B.A., Saunders M.A. (1993): *MINOS 5.4 User's guide*, SOL, Department of Operations Research, Stanford University, Technical Report SOL 83-20R

[19] Pinter J.D. (1996): *Global Optimization in Action*, Kluwer Academic Publishers, Dordrecht

[20] Pinter J.D. (2001): *LGO: An integrated application development and solver program system for continuous global optimization*, User Manual, Pinter Consulting Services, Halifax, Canada

[21] 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, Heidelberg

[22] Rastrigin L.A. (1974): *Systems of Extremal Control*, Nauka, Moscow (in Russian)

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

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

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

[26] Schittkowski K. (2002): *Test problems for nonlinear programming - user's guide*, Report, Department of Mathematics, University of Bayreuth

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

[28] Schittkowski K. (2002): *Test problems for nonlinear programming - user's guide*, Report, Department of Mathematics, University of Bayreuth

[29] 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

[30] Storn R., Price K. (1997): *Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization, Vol. 11, 341-359

[31] Strongin R.G. (1978): *Numerical Methods of Multiextremal Optimization*, Nauka, Moscow (in Russian)

[32] Törn A., Zilinskas A. (1989): *Global Optimization*, Lecture Notes in Computer Science, Vol. 350, Springer, Heidelberg

[33] Trefethen N. (2002): *A hundred-dollar hundred-digit challenge*, SIAM News, Vol. 35., 1

[34] Wang G.G., Dong Z., Aitchison P. (1999): *Adaptive response surface method - a global optimization scheme for approximation-based design problems*, Report, Department of Mechanical and Industrial Engineering, University of Manitoba, Winnipeg, Canada

[35] Zabinsky Z.B., Graesser D.L., Tuttle M.E., Kim G.I. (1992): *Global optimization of composite laminates using improving hit and run*, in: Floudas C. and Pardalos P. (eds.),*Recent Advances in Global Optimization*, Princeton University Press, 343-368