A local and globalized, constrained and simple bounded Nelder-Mead method

Sebastian Wolff

2004/10/20

Abstract

An algorithm for solving nonlinear, multimodal and discontinuous local optimization problems is described. The Nelder-Mead method for direct optimization, wherein derivatives are not possible or to expensive to determine, has been extended to simple bounds and nonlinear equality and inequality constraints, the latter handled by an adaptive linear penalty (BNM). To globalize the method, probabilistic restarts are introduced utilizing a memory of previous iterations. It is attempted to detect failures of the local optimizers through reinitializing the simplex. The resulting method (GBNM) is particularly suitable for engineering problems, where the computational costs, e.g. the number of objective function evaluations, should not exceed a certain limit. Both algorithms were implemented in ANSI C/C++ and have been published under the open source license LGPL, version 2. The programming interface is explained.

1 Introduction

In this paper a global, direct algorithm for solving nonlinear optimization problems with simple bounds and nonlinear constraints is described. The problem can be written as

$$\min_x f(x)$$

subjected to

$$l \leq x \leq u, g_i(x) \leq 0, i = 1 \ldots ne, h_i(x) = 0, i = (ne + 1) \ldots m$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function whose gradient is not available, the vectors $l$ and $u$ represent lower and upper bounds on the variables, $n$ the number of variables, $g_i$ and $h_i$ the inequality and equality constraints with a total number of constraints $m$ and a number of equality constraints $ne$. The objective function might be non-convex and might have multiple local minima.

The Nelder-Mead method [6] has already been proposed in 1965. It is a simple, intuitive and relatively stable method that approaches the optimum in
great steps in the beginning of the search and that can be applied to discontinuous problems. Though, no statements on convergence behaviour can be made. Moreover, if the method fails for certain reasons, the Karush-Kuhn-Tucker conditions can not be applied to verify if the method failed or succeeded.

Various direct methods have been developed. Based on the idea of the presented algorithm, multidirectional search methods were implemented supporting parallel computations. Other methods are, for example, pattern search algorithms, the method of conjugated directions and the great class of genetic algorithms.

2 Algorithm

2.1 Basic principles

The Nelder-Mead method constructs a simplex of \( n+1 \) vertices. There are three construction principles to determine a new point. Let \( S \subset \mathbb{R}^n \) be a simplex with the \( n+1 \) vertices \( x_0, \ldots, x_n \), and

\[
s_j = \frac{1}{n} \sum_{i=0, i \neq j}^{n} x_i
\]

denotes the center of gravity with respect to \( x_j \).

**Reflexion** of the corner \( x_j \) at \( s_j \). The new point is determined from

\[
x^r = s_j + r(s_j - x_j)
\]

with the reflexion constant \( 0 < r \leq 1 \).

![Figure 1: reflexion](image)

**Expansion** of the corner \( x^r \) in direction \( s_j - x_j \). The new point is determined from

\[
x^e = s_j + \gamma(x^r - s_j)
\]
with the expansion constant $\gamma > 1$.

**Figure 2: expansion**

**Contraction** whereby three different types can be distinguished. $0 < \beta < 1$ denotes a contraction constant.

1. Partial interior contraction of $x^j$ in direction $s^j - x^j$. The new point is determined from

   $x^c = s^j + \beta(x^j - s^j)$

   **Figure 3: partial interior contraction**

2. Partial exterior contraction of $x^r$ in direction $s^j - x^r$. The new point is determined from

   $x^c = s^j + \beta(x^r - s^j)$

3
3. Total contraction to \( x^j \). All points will be replaced by

\[
x^i = \frac{1}{2} (x^i - x^j)
\]

2.2 A local, bounded Nelder-Mead method (BNM)

With these prefaces a modification of Nelder-Mead can be explained. A point \( x^{(k,i)} \) denotes the \( i \)-th vertex of the simplex at the \( k \)-th iteration.

1. Select an initial point \( x^{(0,0)} \) and determine the vertices of the initial simplex \( S_0 \) of size \( a \)

\[
x^{(0,i)} = x^{(0,0)} + pe_i + \sum_{k = 1}^{n} qe_k, \ i = 1 \ldots n
\]

where \( e_i \) are the unit base vectors and

\[
p = \frac{a}{n\sqrt{2}}(\sqrt{n+1} + n - 1)
\]

\[
q = \frac{a}{n\sqrt{2}}(\sqrt{n+1} - 1)
\]

see Haftka et al [3]. Set \( k = 0 \).

2. Determine the simplex point with highest objective function value \( x^{(k,h)} \), the simplex point with second largest function value \( x^{(k,s)} \), the vertex with smallest function value \( x^{(k,l)} \), the centroid simplex point with respect to \( x^{(k,h)} \), named \( s^{(k,m)} \), and the corresponding objective function values \( f^{(k,h)} = f(x^{(k,h)}) \), \( f^{(k,s)} = f(x^{(k,s)}) \), \( f^{(k,l)} = f(x^{(k,l)}) \).

3. Apply a reflation with respect to \( x^{(k,h)} \)

\[
x^r = x^{(k,m)} + r(x^{(k,m)} - x^{(k,h)})
\]

Project its coordinates on the bounds, if \( x^r \) is out of the domain.
4. We distinguish between three cases:

(a) $f(x^r) < f^{(k,l)}$, which means that the reflection created a new minimum. It is attempted to get an even better point through expansion of $x^r$ in direction of $x^r - s^{(k,m)}$

$$x^c = s^{(k,m)} + \gamma (x^r - s^{(k,m)})$$

If $x^c$ is out of the domain its coordinates are projected on the bounds. $x^{(k,h)}$ will be replaced by the better of both points $x^r$ or $x^c$:

$$x^{(k+1,h)} = \begin{cases} x^c, & f(x^c) < f(x^r) \\ x^r, & \text{else} \end{cases}$$

(b) If $f(x^r) \leq f^{(k,s)}$ $x^r$ will not be worse than all vertices except $x^{(k,l)}$, usually better than $x^{(k,h)}$ which will be replaced:

$$x^{(k+1,h)} = x^r$$

(c) If $f(x^r) \geq f^{(x^{(k,h)})}$ it was probably wrong to do the reflection along the selected direction. The opposite direction is attempted. An interior contraction from $x^{(k,h)}$ in direction $s^{(k,m)} - x^{(k,h)}$ will be applied, yielding

$$x^c = s^{(k,m)} + \beta (x^{(k,h)} - s^{(k,m)})$$

Else, if $f(x^r) < f^{(x^{(k,h)})}$ the selected direction might have been right. But since all vertices except $x^{(k,h)}$ are better than $x^r$ it can be concluded to go closer to the simplex again. An exterior contraction from $x^r$ in direction $s^{(k,m)} - x^r$ is performed, that means $x^{(k,h)}$ will be replaced by $x^r$ and the previous contraction equation is applied. After the partial contraction, if $f(x^c) < f^{(x^{(k,h)})}$ the resulting point has been improved. $x^{(k,h)}$ will be replaced through

$$x^{(k+1,h)} = x^c$$

Otherwise, if $f(x^c) \geq f^{(x^{(k,h)})}$ all attempts getting any improvement failed. A total contraction with respect to $x^{(k,l)}$ will be performed. For $i \neq l$ we set

$$x^{(k+1,i)} = \frac{1}{2}(x^{(k,i)} + x^{(k,l)}), \quad i = 0 \ldots n$$

5. Set $k := k + 1$.

6. Check convergence. Since no suppositions have been made for the objective function $f$, it is not easy to define an appropriate termination criterion. Nelder and Mead proposed that the standard deviation of the
objective function values of the simplex points must be smaller than a certain tolerance constant \( \epsilon > 0 \). If the condition
\[
\left( \frac{1}{n+1} \sum_{i=0}^{n} f(x^{(k,i)}) - \bar{f}_k \right)^\frac{1}{2} < \epsilon
\]
with
\[
\bar{f}_k = \frac{1}{n+1} \sum_{j=0}^{n} f(x^{(k,j)})
\]
is true terminate the iteration and goto (7), otherwise goto (2).

7. End.

The vertex \( x^{(k,l)} \) can be seen as the current iteration point at iteration \( k \). In the presented implementation the current iteration will be finished and the convergence test will be done, even in the case that the maximum allowed number of objective function calls has been reached during the last iteration.

A search direction and a step size are not explicitly determined. The strategy simply assures that
\[
f(x^{(k+1,l)}) \leq f(x^{(k,l)})
\]

### 2.3 An adaptive linear penalty function

Several methods for handling constraints utilizing originally unconstrained optimizers have been invented. Some approaches use sequential linear or quadratic approximations of the constraints. Others modify the objective function in such a way that unconstraint methods can be applied and that all saddle points of the modified problem are solutions of the original task. Barrier methods add barrier terms to the objective function, which will grow to infinity if the current design point approaches the edge of the constrained domain. As a result the design point will never leave the feasible domain. Penalty methods add penalty terms to the objective function. The primal problem,
\[
\min_x f(x)
\]
\[l \leq x \leq u, g_i(x) \leq 0, \quad i = 1 \ldots ne, h_i(x) = 0, \quad i = (ne + 1) \ldots m\]
is rewritten in an unconstrained penalized form,
\[
\min_x L(x, \lambda), \quad \text{where}
\]
\[L(x, \lambda^k) = f(x) + \sum_{i=1}^{m} \lambda_i^k P_i(x)\]
where \( L \) denotes the penalty function, \( P_k \) relative penalty costs and \( \lambda_k \) a series of the vectors of the penalty parameters with
\[
\bullet \quad P_i(x) \begin{cases} = 0, & \text{if } x \text{ in the interior} \\ > 0 & \text{outwards} \end{cases}
\]
\[ \lambda^{k+1} > \lambda^k \geq 0, \quad k \in \mathbb{N}, \quad \lim_{k \to \infty} \lambda^k = \infty \]

Through the demand \( \lambda^k \to \infty \) and the unlimited growth of the penalty costs the constraints \( g(x) \leq 0 \) and \( h(x) = 0 \) are less violated for growing \( k \). Since a direct method is discussed, derivatives are not used at all and the Lagrangian \( L \) does not need to be differentiable at places, where \( g(x) = 0 \) or \( h(x) = 0 \). Therefore, a linear penalty can be used, yielding

\[
P_i(x) = \begin{cases} 
\max(0, |h_i(x)|), & \text{if } i \in [1 \ldots ne] \\
\max(0, g_i(x)), & \text{if } i \in [(ne + 1) \ldots m]
\end{cases}
\]

Appropriate values of the penalty parameters \( \lambda_i \) need to be estimated. They are initialized as 0 and will be updated after each determination of a new simplex point \( x^{(k+1,j)} \) by the BNM algorithm. The penalty parameters will be increased if the related constraints of the new or, respectively, changed simplex point \( x^{\text{new}} \) are violated. Moreover, only in the case that the new simplex point would be the vertex with smallest objective (lagrangian) function value \( L(x^{\text{new}}) \) among all other points, the penalty parameters will be changed:

1. Determine the vertex with smallest Lagrangian

\[ x^\text{best} = \arg \left( \min_{x \in x^{\text{new}}, x^\text{best}, \text{simplex vertices}} L(x, \lambda^k) \right) \]

2. If \( (L(x^{\text{new}}, \lambda^k) \leq L(x^\text{best}, \lambda^k)) \) set \( \lambda_i^{k+1} \)

\[ \lambda_i^{k+1} = \lambda_i^k + s \times P_i(x), \quad i = 1 \ldots m \]

where \( s \) is a positive step size.

A large step size \( s \) might let BNM fail in creating a well designed simplex for the case that the penalty function values and, therefore, the hierarchy among the simplex points changes with every update of \( \lambda \). A small step size, paired with a great termination tolerance \( \epsilon \), might increase the penalty parameters not fast enough, such that the optimum will be found in a region with strongly violated constraints or such that it takes too many iterations. Hence, the step size could be a series as well, which will be a constant small value and a large one for certain iteration numbers. Let \( k \) denote the current iteration, \( s_s \) the standard step size and \( s_L \) a step size for sudden augmentation with \( 0 < s_s < s_L \). The penalty parameters will be strongly augmented either once after a certain number of iterations with \( k = k_S > 0 \) or within regular time intervals for \( k_S < 0 \)

\[ s_k = \begin{cases} 
s_L, & \text{if } (k_S > 0) \text{ and } (k = k_S) \\
s_L, & \text{if } (k_S < 0) \text{ and } (k \mod |k_S| = 0) \\
s_s, & \text{else}
\end{cases} \]

Another way (Luersen at al [5]) would have been to stop updating the penalty parameters after, for example, 100 iterations. This would decouple the primal and dual searches. Considering generalized lagrangian theory the problem could then be expressed as

\[ f(x^*) = \min_x f(x) = \min_{x, \lambda} \max L(x, \lambda) = \max_{\lambda} \min_x L(x, \lambda) \]
2.4 Globalization of a local optimizer

To globalize a local optimizer either problem-specific knowledge, as long as it exists, will be added to the search or local and global methods are mixed. The simplest strategy is to link the local searches in a series. Firstly, a rough global optimization of strongly limited cost is performed which solution will be refined by a local optimizer.

Another simple strategy is to find well defined starting points for the local method. This can be done by restarting the local search probabilistically and terminating local searches if they approach regions which have been already explored. Alternatively, the bounded domain can be partitioned. For each cell of the grid a representative or randomly determined point will be used as a starting point of the local optimizer which will be terminated if the search leaves the current stratum. The advantage of the grid over a purely random restart is that the domain is covered equally. Its disadvantage is that the computational costs of every search in one stratum are unknown before starting the global search. In the following paragraphs a probabilistic restart (Luersen et al. [5]) will be presented which is right in between both approaches and which tries to maximize the probability of finding the global minimum for a given maximum computational cost.

2.4.1 Probabilistic restart using a memory of previous iterations

Let us assume to have generated a list of \( N \) initial points \( x^i \). The basic approach is to reduce the probability that the next initial point is close to a prior initial point. This is done through associating a symmetrical probability distribution for each saved point, for example the Gaussian distribution - assuming having no correlation - represented by its probability density for the saved point \( x^i \)

\[
p^i(x) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi} \sigma_{X_j}} \exp \left( -\frac{1}{2} \left( \frac{x_j - \bar{X}_j}{\sigma_{X_j}} \right)^2 \right)
\]

wherein the means \( \bar{X}_j \) are identical with the saved points \( x^i \), and \( n \) is the dimension. The variances \( \sigma^2_j \) are estimated from

\[
\sigma^2_j = \alpha (x_j^{max} - x_j^{min})
\]

\( \alpha \) is a positive parameter that controls the length of the Gaussians. \( x_j^{max} \) and \( x_j^{min} \) are the bounds in the \( j \)-th direction. For simplicity, the variances are kept constant.

The probability \( p(x) \) of sampling an already saved point \( x^i \) can be expressed through

\[
p(x) = \frac{1}{N} \sum_{i=1}^{N} p^i(x)
\]
The probability density is such that \( \int_{-\infty}^{\infty} p(x)dx = 1 \), but since a bounded domain \( \Omega \) is considered, a bounded probability must be introduced, yielding

\[
\tilde{p}(x) = \frac{p(x)}{M}, \quad M = \int_{\Omega} p(x)dx
\]

Let \( \phi(x) \) denote the probability density of sampling the next initial point. Sampling the point \( x_H = \text{arg}(\max_x \tilde{p}(x)) \) as the next initial point is assumed to be impossible. Therefore, the probability density \( \phi(x) \) is determined from

\[
\phi(x) = \frac{H - \tilde{p}(x)}{\int_{\Omega} (H - \tilde{p}(x))dx}
\]

\[
H = \max_{x \in \Omega} \tilde{p}(x)
\]

\( \phi \) is approximated by a Gaussian distribution, following the previously described scheme. The mean value \( \overline{x}^{N+1} \) is determined from solving the maximization problem, which can be replaced by

\[
\overline{x}^{N+1} = \arg\min_{x \in \Omega} p(x)
\]

As a result, the next starting point will be \( x^{N+1} = \overline{x}^{N+1} \). For simplicity, \( p(x) \) is minimized through generating \( N_r \) random vectors \( x_k \) with \( x^{N+1} = \arg\min_{k} p(x_k) \). Saved are not only initial points, but also found local optima. The starting and local convergence points efficiently summarize the topology of the basins of attraction.

### 2.5 A saver Nelder-Mead search

Since the Karush-Kuhn-Tucker conditions are not applicable, Luersen et al [5] introduced a set of restart options and three convergence criteria to detect if BNM has failed or, respectively, succeeded.

1. Probabilistic restarts (see 2.4.1) repeat local searches until a fixed total cost, \( C_{\text{max}} \), has been reached. The probability of having located a global optimum increases with the number of probabilistic restarts. Luersen et al used a uniformly distributed random variable between 2 and 10% of the smallest domain dimension for the initial simplex size \( a \).

2. If a possible optimum was found, but the optimality is uncertain, the simplex will be reinitialized with a smaller simplex size \( a_S \) (small test or optimality check). Returning to the same point implies an optimum.

3. If the simplex has degenerated a large test will be performed, meaning reinitializing the simplex with a large simplex size \( a_L \).

The BNM convergence check will be replaced by terminating the local search if one of the following conditions is satisfied:
1. A simplex is small if
\[
\max_{k=1,\ldots,n+1} \left( \sum_{i=1}^{n} \left| \frac{e^k_i}{x_i^{\max} - x_i^{\min}} \right| \right) < \epsilon_{s1}
\]
where \(e^k_i\) is the \(i\)-th component of the \(k\)-th edge, \(x_i^{\max}\) and \(x_i^{\min}\) are the bounds in the \(i\)-th direction. Having a small simplex implies convergence. If the point touches the bounds, more tests have to be performed.

2. A simplex is flat if
\[
|f_H - f_L| < \epsilon_{s2}
\]
wherein \(f_H\) and \(f_L\) denote the highest and the lowest objective function values. A flat simplex induced failure and a probabilistic restart. It can appear if the simplex moves into a region of equal objective function values, which can’t be left anymore.

3. A simplex is degenerated if it is neither small, nor touches the bounds, and one of the following conditions is covered:
\[
\min_{k=1,n} \|e^k\| < \epsilon_{s3} \\
\max_{k=1,n} \|e^k\| < \epsilon_{s4} \\
\det(e) \prod_k \|e^k\| < \epsilon_{s4}
\]
where \(e^k\) is the \(k\)-th edge vector, and \(e\) is the edge matrix. This is the most common symptom of a failed Nelder-Mead search (Wright [7]). It describes the case that the simplex has collapsed into a subspace of the search domain which it can not escape.

Since a small test will never exactly return to the same point, a point \(x\) is assumed to equal a point \(x^j\) if the following condition is satisfied:
\[
\left( \frac{1}{n} \sum_{i=1}^{n} (x_i - x^j_i)^2 \right)^{1/2} < \epsilon_{s5}
\]

### 2.6 GBNM

1. Initialization, set \(\text{RESTART} = \text{PROBABILISTIC}\).
2. Do a restart, either \(\text{PROBABILISTIC, SMALLTEST or LARGETEST}\). Save the initial point.
3. One BNM iteration.
4. If the computational cost \(C_{\text{max}}\) has been reached the global search will be terminated. Goto (6)
5. If the convergence criteria (small or flat or degenerated simplex) are not satisfied goto (3). Else we distinguish the following cases:

(a) Is the point an already known optimum? Replace the saved point by the current local optimum, if it is better, set \( \text{RESTART} = \text{PROBABILISTIC} \) and goto (2).

(b) If the simplex is flat, set \( \text{RESTART} = \text{PROBABILISTIC} \) and goto (2).

(c) If it a small or large test is currently performed and we return to the same point, or, if a large test or a probabilistic restart is currently performed and the point does not touch the bounds and the simplex is small, the found point is a local optimum. Save the optimum, set \( \text{RESTART} = \text{PROBABILISTIC} \) and goto (2).

(d) If \( \text{LARGE TEST} \) or \( \text{PROBABILISTIC} \) and we do not return to the same point and the point is on the bounds, we might have found an optimum. To verify it, set \( \text{RESTART} = \text{SMALL TEST} \) and goto (2).

(e) If a small test is running, the point is not on the bounds and we did not return to the same point, a local optimum might have been found. Save the optimum, set \( \text{RESTART} = \text{PROBABILISTIC} \) and goto (2).

(f) If no small test is running, the point is not on the bounds, the simplex is not small and the simplex is degenerated, set \( \text{RESTART} = \text{LARGE TEST} \) and goto (2).

(g) Else, set \( \text{RESTART} = \text{SMALL TEST} \) and goto (2).

6. Terminating the global search. The best simplex vertex of the current local search will be saved as a possible local optimum. The global optimum is selected from \( \min_{j=1}^{N_{opt}} f(x^j) \) with the number of saved local optima \( N_{opt} \).

**Note!** Since we experienced difficulties in finding optima at all using the GBNM proposed by Luersen et al, the method has been changed in the convergence check (step 5). Instead, the convergence check, proposed by Nelder and Mead is taken. The additional three tests on flatness, smallness and degeneracy are performed like stated above.

### 3 Numerical Results

#### 3.1 Parameter choice

For BNM, empirical analysis has shown that constants \( 0.4 \leq \beta \leq 0.6, \ 2 \leq \gamma \leq 3 \) and \( r = 1 \) yield good results. Other constants, namely the step size parameters \( s_s, s_L, k_S \) and the initial simplex size depend on the considered problem.
For GBNM, Luersen et al recommend a Gaussian length parameter \( \alpha = 0.01 \) and a number of random points \( N_r \), used to optimize the probabilistic restart, of \( N_r = 10 \). The larger \( N_r \), the more perfect the pattern created by the initial points. For a small \( N_r \), a biased random reinitialization will be performed. The choice of \( N_r \) should depend on the test function and the problem dimension.

### 3.2 Test functions, BNM

Two-dimensional analytical test functions are considered. For all test cases the following parameters are set: the bounds \( l^T = (-5, -5), u^T = (5, 5) \), the initial design \( x^T = (0, 0) \), \( a = 0.1 \), accuracy \( \epsilon = 10^{-4} \), step sizes \( s_s = 0.1, s_L = 10, k_S = 10 \), the operation parameters \( r = 1, \beta = 0.5, \gamma = 2 \).

1. **Rosenbrock**:
   \[
   \min_x 100(x_2 - x_1^2)^2 + (1 - x_1)^2
   \]
   The minimum is 0 at (1, 1). After 42 iterations and 83 function calls the minimum \( 6.6 \times 10^{-5} \) was found at (1.0044, 1.0095).

2. **Bazaraa/Shetty**:
   \[
   \min_x (x_1 - 2)^4 + (x_1 - 2x_2)^2
   \]
   The minimum is 0 at (2, 1). After 27 iterations and 54 function calls the minimum \( 3.74 \times 10^{-5} \) was found at (2.0596, 1.0323).

3. **Himmelblau**:
   \[
   \max_x (x_1^2 + x_2 - 11)^2 + (x_1 - x_2^2 - 7)^2
   \]
   The function has 4 minima of 0 and one maximum at \((-0.270845, -0.923039)\). It was found after 21 iterations and 40 function calls with an objective function value of 181.62 at \((-0.27140, -0.91946)\).

4. **Rosenbrock, 1 constraint**:
   \[
   \begin{cases}
   \min_x 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\
   \text{such that } 4 - x_1^2 = 0
   \end{cases}
   \]
   The minimum is 1 at (2, 4). After 69 iterations and 130 function calls the minimum 1.0001 was found at (2.0000, 4.0010).

### 3.3 Test functions, GBNM

For all test cases the following parameters are set: \( a = 0.01, a_{Small} = 0.001 \), step sizes \( s_s = 0.1, s_L = 10, k_S = 30 \), the operation parameters \( r = 1, \beta = 0.5, \gamma = 2 \). The accuracies are set to \( \epsilon = 0.00001, \epsilon_{s1} = 10^{-7}, \epsilon_{s2} = 10^{-10}, \epsilon_{s3} = 10^{-15}, \epsilon_{s4} = 10^{-15}, \epsilon_{s5} = 10^{-3} \).

1. **Himmelblau**:
   \[
   \min_{x \in [-5,5]} (x_1^2 + x_2 - 11)^2 + (x_1 - x_2^2 - 7)^2
   \]
   The function has 4 minima with \( f(x) = 0 \) and one maximum at \((-0.270845, -0.923039)\). The global optimization was performed 50 times with a number of function calls equal to 1000. All four minima were always found. Their
coordinates are on average: (3.000087, 1.999645), (3.584278, -1.847779), (-3.779150, -3.283327), (-2.805129, 3.131457).

2. Michalewicz and Schoenauer (1):

\[
\begin{align*}
\min_{x_i \in [0.001, 20]} & -\frac{\sin(2\pi x_1)\sin(2\pi x_2)}{x_1(x_1+x_2)} \\
\text{such that } & g_1(x_1, x_2) = x_1^2 - x_2 + 1 \leq 0 \\
& g_2(x_1, x_2) = 1 - x_1 + (x_2 - 4)^2 \leq 0 \\
\end{align*}
\]

The initial value was set to (10, 10). The problem has an optimum of -0.0958248 at (1.228, 4.245). Using 5000 function calls and performing the optimization 50 times, the global optimum was always found. Though, in 15 optimizations the lowest of the saved local optima violated the constraints. This had to be checked manually.

3. Michalewicz and Schoenauer (2):

\[
\begin{align*}
\min_{x_i \in [-20, 20]} & (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11) + \\
& 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \\
\text{such that } & -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\
& -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\
& -196 + 23x_1 + x_2^2 + 6x_3^2 - 8x_7 \leq 0 \\
& 4x_1^2 + x_2^4 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \\
\end{align*}
\]

The global minimum is 680.6300573 at (2.330499, 1.951372, -0.4775414, 4.365726, -0.624487, 1.038131, 1.594227), with the first and last constraints active. After 500 evaluations a value of 752.064 has been found at (1.276966, 2.095535, 0.382304, 4.015710, 0.172317, -1.521929, 1.828635). After 3000 function calls an optimum was found at (1.124591, 1.817193, 0.410052, 1.817193, 0.410052, 1.817193, 0.410052) and \( f(x^*) = 698.450 \). Further increments of the number of function calls do not yield a closer optimum. It is suggested to tune the parameters to get better results.

### 4 Implementation of BNM

BNM has been implemented in the software package SLang (http://www.uni-weimar.de/Bauing/ism/engl.Version/SLang/slang.Short.html). In addition, the base package can be downloaded from http://www.uni-weimar.de/~wol3. The downloadable package is licensed under the LGPL, version 2. It is written in C++. If you want to convert it to ANSI C, you probably only need to remove the line comments ("/**/comment") and other C++ niceties like “inline” operators or “const” parameters.

**Files:** The package consists of the files optimize_bnm_.h and optimize_bnm_.cpp. The header contains inline functions that are used by the GBNM algorithm as well. If you copy, distribute, modify or use the software, please refer to the conditions outlined in LICENSE.LIB.
4.1 General syntax:

```c
void optimize_bnm
    ( int * ifail, int * task, int n, double * design, double * objective,
      double * simplex, double * simplex_f,
      double * Pr, double * fr, double * lr, double * Pm,
      double a, double r, double beta, double gamma,
      double eps, int n_eq, int n_meq, double * constr,
      double step_size, double * constr_mat, double * lambda,
      int augment_interval, int * augment_tmp, double augment_size,
      double * constr_fr, double * lagrangian,
      double * lb, double * ub, int * tb,
      int check_initial_consistency );
```

`ifail` reverse communication flag

<table>
<thead>
<tr>
<th>in/out:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>converged</td>
</tr>
<tr>
<td>-1</td>
<td>user must update the objective value and the constraints at new design point</td>
</tr>
<tr>
<td>-2</td>
<td>user must update the objective gradients (not possible here, for other methods)</td>
</tr>
<tr>
<td>-3</td>
<td>user must do nothing, just call it again...</td>
</tr>
<tr>
<td>&gt;0</td>
<td>an error occurred</td>
</tr>
<tr>
<td>1</td>
<td>not enough memory</td>
</tr>
<tr>
<td>2</td>
<td>initial simplex out of bounds, reduce simplex size a or change initial design point</td>
</tr>
<tr>
<td>3</td>
<td>invalid parameters</td>
</tr>
</tbody>
</table>

`task` a kind of memory, saves information which tell the function how to proceed when called another time

<table>
<thead>
<tr>
<th>in: initial value = -1, out:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>-1</td>
<td>means start, design contains the initial point</td>
</tr>
<tr>
<td>-2</td>
<td>start a new iteration</td>
</tr>
<tr>
<td>-3</td>
<td>reflexion has been done, go on</td>
</tr>
<tr>
<td>-4</td>
<td>expansion has been done, go on</td>
</tr>
<tr>
<td>-5</td>
<td>contraction has been done, go on</td>
</tr>
<tr>
<td>-6</td>
<td>go on with checking convergence</td>
</tr>
<tr>
<td>0</td>
<td>converged, do nothing</td>
</tr>
<tr>
<td>&gt;0</td>
<td>the vertex index of the current simplex, this is the difference to task=1..n is that this is a result during the iteration and not</td>
</tr>
<tr>
<td>&gt;n+1</td>
<td>the vertex index of the current simplex -(n+1), this is the current 'design' point for the difference to task=1..n is that this is a result during the iteration and not the difference to task=1..n is that this is a result during the iteration and not immediately after start. The iteration will continue with task==6 afterwards</td>
</tr>
</tbody>
</table>

- \( n \) int, number of variables
- design \( R(n) \), in: initial design, out: vertex with index 'task' if task > 0
- objective in: objective function at design
- simplex \( R(n,n+1) \), all \((n+1)\) vertexes of the simplex
- simplex_f \( R(n+1) \), function values at the simplex vertexes
- fr \( R(1) \), function value at pr, that means at the vertex we got after the reflexion, internal
- lr \( R(1) \), corresponding lagrangian, internal
- Pr \( R(n) \), see fr, internal
- Pm \( R(n) \), internal use only - center of the simplex
- a \( R(1) \), size of the initial simplex
- eps \( R(1) \), eps > 0, termination criteria for convergence
- lb \( R(n) \), lower bounds for design variables
- ub \( R(n) \), upper bounds for design variables
- tb \( \text{int}(n) \), defines boundary type:

<table>
<thead>
<tr>
<th>tb[i]</th>
<th>x[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>unbounded</td>
</tr>
<tr>
<td>1</td>
<td>has only a lower bound</td>
</tr>
<tr>
<td>2</td>
<td>has both lower and upper bounds</td>
</tr>
<tr>
<td>3</td>
<td>has only an upper bound</td>
</tr>
</tbody>
</table>
r       R(1), reflection coefficient
beta    R(1), contraction coefficient
gamma   R(1), expansion coefficient
n_eq    int(1), number of equality constraints, ≥ 0
n_meq   int(1), number of constraints, equals number of inequality constraints + n_eq, ≥ 0
constr  R(n_meq), constraints function values
         \[ \begin{array}{c|c} 
         i & \text{constr}[i] \\
         \hline
         0 \ldots n_{eq} - 1 & h_i(\text{design}) = 0 \\
         n_{eq} \ldots n_{meq} - 1 & g_i(\text{design}) \leq 0 \\
         \end{array} \]
step_size R(1), >0, step size for the optimization problem to determine constraint parameters \( s_s \)
augment_interval int(1), if \( \lambda \)'s are updated the augment_interval'th time they are increased, \( k_S \)
         \[ \begin{array}{c|c}
         0 & \text{augment option disabled} \\
         <0 & \text{it will be augmented ONCE only} \\
         >0 & \text{it will be augmented at the end of each interval} \\
         \end{array} \]
augment_size R(1), >step_size, the step_size that is used for at augment_interval, \( \varepsilon_L \)
augment_tmp int(1), current augment counter, internal, \( k \)
lambdaxa R(n_meq), constraint parameters, internal
constr_mat R(n_meq,n+1), constraints for all simplex vertices, internal
constr_fr  R(n_meq), constraints for \( c_r \), internal
lagrangian R(n+1), lagrangian function values
check_initial_consistency int(1). If not '0' the algorithm checks that the initial simplex will be in the bounded domain. If so fail will return an error.

4.2 Driver implementation

It is advisable to implement a framework which only needs changes of the parameters and of the objective function values. Fragments of the SLang implementation are presented in the file BNM_EXAMPLE. Their purpose is to show how such an implementation could principally look like.
Outline of the interior loop:
1. if (ifail <= 0)
   (a) if (ifail == -1) increase number of function calls
   (b) if (task == -2) increase number of iterations ($k$)
   (c) call BNM
   (d) check if any errors appeared (ifail > 0)
   (e) check if maximum number of function calls has been reached, determine the currently best simplex point
   (f) check if maximum number of iterations has been reached, determine the currently best simplex point
   (g) while (ifail == -3) goto (1).

Outline of the exterior loop:
1. Set parameters.
2. Set initial design and save it in 'design'.
3. Set objective and constraints for initial design.
4. Set task = -1, ifail = 0
5. Do
   (a) if (ifail == -1) update objective and constraints for 'design'
   (b) call interior loop
   (c) while (ifail < 0) goto (5)
6. If (ifail == 0) probably found an optimum, saved in design/objective.

5 Implementation of GBNM

GBNM has been implemented in the software package SLang (http://www.uni-weimar.de/Bauing/ism/engl.Version/SLang/slang.Short.html). In addition, the base package can be downloaded from http://www.uni-weimar.de/~wolff3. The downloadable package is licensed under the LGPL, version 2. It is written in C++. If you want to convert it to ANSI C, you probably only need to remove the line comments (“//comment”) and other C++ niceties like “inline” operators or “const” parameters.

To make it compilable, you need to provide the following functions:

- int sec_since_midnight();
  returns the current age of the day in seconds. used for ’srand()’.
• `double compiler_generator();`
  returns a uniformly distributed random number between 0 and 1.

• `void optimize_gbnm_determinant(double * matrix, double * det, int n);`
  computes the determinant of a $n \times n$ matrix and stores the result in `det`.

**Files:** GBNM is using the basic BNM method which consists of the files `optimize_bnm_.h` and `optimize_bnm_.cpp`. The GBNM algorithm can be found in `optimize_gbnm_.cpp`. If you copy, distribute, modify or use the software, please refer to the conditions outlined in LICENSE.LIB.

### 5.1 General syntax:

```c
void optimize_gbnm
  ( int * ifail, int *task, int *start_mode, int *restart_save,
    double * design, double * objective, int * num_objective_call,
    int num_max_objective_call, int * num_sub_objective_call,
    int num_max_sub_objective_call, int n, int Nr,
    int n_eq, int n_meq, double * constr,
    double step_size, double * constr_mat, double * lambda,
    int augment_interval, int * augment_tmp, double augment_size,
    int N, double * pt, double alpha, double a_small, double a_large,
    int Nfail, double * fail_d, int * fail_mode,
    double * simplex, double * simplex_f,
    double * Pr, double * fr, double * lr,
    double * Pm, double * constr_fr, double * lagrangian,
    double * a, double r, double beta, double gamma, double accuracy
  );
```

`ifail` reverse communication flag
`in/out:`
converged

-1 user must update the objective value and the constraints at new design point

-2 user must update the objective gradients (not possible here, for other methods)

-3 user must do nothing, just call it again...

-4 user must allocate another column in opt_d and opt_f and must increase Nopt by 1

-5 the user must allocate another column in pt and must increase N by 1

-6 the user must allocate another column in fail_d and fail_mode, and must increase Nfail by 1

>0 an error occurred

1 not enough memory

2 initial simplex out of bounds, reduce simplex size a or change initial design point

3 invalid parameters

<table>
<thead>
<tr>
<th>task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>means start, design contains the initial point</td>
</tr>
<tr>
<td>-2</td>
<td>start a new iteration</td>
</tr>
<tr>
<td>-3</td>
<td>reflection has been done, go on</td>
</tr>
<tr>
<td>-4</td>
<td>expansion has been done, go on</td>
</tr>
<tr>
<td>-5</td>
<td>contraction has been done, go on</td>
</tr>
<tr>
<td>-6</td>
<td>go on with checking convergence</td>
</tr>
<tr>
<td>0</td>
<td>converged, do nothing</td>
</tr>
<tr>
<td>&gt;0</td>
<td>the vertex index of the current simplex, this is the difference to task=1..n is that this is a result during the iteration and not</td>
</tr>
<tr>
<td>&gt;n+1</td>
<td>the vertex index of the current simplex -(n+1), this is the current &quot;design&quot; point for the difference to task=1..n is that this is a result during the iteration and not the difference to task=1..n is that this is a result during the iteration and not immediately after start. The iteration will continue with task=-6 afterwards</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start_mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>finished, reached final number of iterations or evaluations</td>
</tr>
<tr>
<td>-1</td>
<td>START (initial value)</td>
</tr>
<tr>
<td>-2</td>
<td>PROBABILISTIC RESTART</td>
</tr>
<tr>
<td>-3</td>
<td>SMALL TEST</td>
</tr>
<tr>
<td>-4</td>
<td>LARGE TEST</td>
</tr>
</tbody>
</table>
restart_save  int, like start_mode, but saves additional internal information for
the current run

n       int, number of variables
Nr      int, number of support points to get optimum start points on restarts, 
        Nr

design  R(n), in: initial design, out: vertex with index 'task' if task > 0

objective in: objective function at design

N       int, in: number of saved points, start with '1' (used for restarts)
pt      R(n,N), out: saved points, used for restarts, start with 1 allocated
        column!

xi      R(n,Nr), must have one column allocated at START

alpha   R, in: Gaussian length parameter

a_small R, initial simplex size for small test, eg. 0.001

a_large R, initial simplex size for large test and probabilistic restart, eg. 0.01

Nopt    int, in: number of saved local optima, start with '0'

opt_d   R(n,Nopt), out: saved local optima (design points), start with 'NULL'
        pointer

opt_f   R(Nopt), out: saved local optima (objective), start with 'NULL'
        pointer

Nfail   int, in: number of saved points fail_d

fail_d  R(n,Nfail), out: design points where a failure or another event occ- 
        cured, start with 'NULL' pointer

fail_mode int(nfail), out: failure mode at the related point, start with 'NULL'
        pointer

<table>
<thead>
<tr>
<th></th>
<th>simplex is flat, START PROBABILISTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>simplex touches bounds (start a SMALL TEST)</td>
</tr>
<tr>
<td>3</td>
<td>simplex has degenerated, start a LARGE TEST</td>
</tr>
<tr>
<td>4</td>
<td>simplex is small (if not small test), START PROBABILISTIC</td>
</tr>
<tr>
<td>5</td>
<td>else: a SMALL TEST was started and NO POSSIBLE optimum found</td>
</tr>
<tr>
<td>6</td>
<td>like 5, but it was already a small test running, do PROBABILISTIC RESTART</td>
</tr>
</tbody>
</table>

x       R(n), saved likely optimum during SMALL/LARGE TEST

edges   R(n,n), edges of simplex, created at end of T1
simplex \( R(n,n+1) \), all \((n+1)\) vertexes of the simplex

simplex_f \( R(n+1) \), function values at the simplex vertexes

fr \( R(1) \), function value at pr, that means at the vertex we got after the reflection, internal

lr \( R(1) \), corresponding lagrangian, internal

Pr \( R(n) \), see fr, internal

Pm \( R(n) \), internal use only - center of the simplex

\( a \) \( R \), internal, current (initial) size of the simplex

num_max_sub_objective_call \( \text{int} \), in: maximum number of function calls during one LOCAL iteration

num_max_objective_call \( \text{int} \), in: maximum number of function calls

num_sub_objective_call \( \text{int} \), in/out: actual number of function calls during a LOCAL iteration

num_objective_call \( \text{int} \), in/out: actual number of function calls

accuracy \( R \), accuracy \( \geq 0 \), termination criteria for convergence of BNM, \( \epsilon \)

\( \text{eps}_s1 \),

\( \text{eps}_s2 \),

\( \text{eps}_s3 \),

\( \text{eps}_s4 \) \( R \), \( \epsilon_s1 \) \( > 0 \), in: termination criterias

\( \text{eps}_s5 \) \( R \), is the tolerance that is used to determine if a point is already known as a local optimum, \( \epsilon_s5 \)

lb \( R(n) \), lower bounds for design variables

ub \( R(n) \), upper bounds for design variables

tb \( \text{int}(n) \), defines boundary type:

<table>
<thead>
<tr>
<th>tb[i]</th>
<th>x[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>unbounded</td>
</tr>
<tr>
<td>1</td>
<td>has only a lower bound</td>
</tr>
<tr>
<td>2</td>
<td>has both lower and upper bounds</td>
</tr>
<tr>
<td>3</td>
<td>has only an upper bound</td>
</tr>
</tbody>
</table>

Note: Should be ‘2’ for all directions. At least ub and lb must be defined.

\( r \) \( R(1) \), reflection coefficient
beta  R(1), contraction coefficient

gamma  R(1), expansion coefficient

n_eq  int(1), number of equality constraints,  \geq 0

n_meq  int(1), number of constraints, equals number of inequality constraints
+ n_eq,  \geq 0

constr  R(n_meq), constraints function values

\begin{array}{|c|c|}
\hline
i  & constr[i] \\
\hline
0 \ldots n_{eq} - 1  & h_i(\text{design}) = 0 \\
n_{eq} \ldots n_{meq} - 1  & g_i(\text{design}) \leq 0 \\
\hline
\end{array}

step_size  R(1),  > 0, step size for the optimization problem to determine con-
straint parameters  s_s

augment_interval  int(1), if  \lambda_s are updated the augment_interval\textsuperscript{th} time they
are increased,  k_S

\begin{array}{|c|c|}
\hline
0  & \text{augment option disabled} \\
<0  & \text{it will be augmented ONCE only} \\
>0  & \text{it will be augmented at the end of each interval} \\
\hline
\end{array}

augment_size  R(1),  > step_size, the step_size that is used for at augment_interval,

augment_tmp  int(1), current augment counter, internal,  k

lambda  R(n_meq), constraint parameters, internal

constr_mat  R(n_meq, n+1), constraints for all simplex vertices, internal

constr_fr  R(n_meq), constraints for cr, internal

lagrangian  R(n+1), lagrangian function values

\subsection*{5.2 Driver implementation}

It is advisable to implement a framework which only needs changes of the pa-
rameters and of the objective function values. Fragments of the SLang imple-
mentation are presented in the file GBNM\_EXAMPLE. Their purpose is to
show how such an implementation could principally look like.

Outline of the interior loop:

1. if (ifail <= 0)

   (a) if (task == -2) increase number of iterations (k)

   (b) call GBNM
(c) If no error occurred, check if maximum number of iterations has been reached. If so, determine the best simplex point:
   i. add a column to \textit{opt\_d} and \textit{opt\_f} and increase \textit{Nopt} by 1.
   ii. set \textit{start\_mode}=200, \textit{ifail}=-5
   iii. call GBNM

(d) check if any errors appeared (\textit{ifail} > 0)

(e) Reverse communication: Listen to commands from GBNM which do not imply an error or updating any function values:
   i. if (\textit{ifail} == -4): allocate memory for a new local optimum: add a column to \textit{opt\_d} and \textit{opt\_f} and increase \textit{Nopt} by 1.
   ii. if (\textit{ifail} == -5): allocate RAM for the probabilistic restart’s memory: add a column to \textit{pt} and increase \textit{N} by 1.
   iii. if (\textit{ifail} == -6): allocate space for a failure point: add a column to \textit{fail\_d} and \textit{fail\_mode} and increase \textit{Nfail} by 1.

(f) while (\textit{ifail} <= -3) goto (1).

Outline of the exterior loop:

1. Set parameters.
2. Set initial design and save it in ‘design’.
3. Set objective and constraints for initial design.
4. Set task = -1, ifail = 0, start\_mode = -1
5. Do
   (a) if (\textit{ifail} == -1) update objective and constraints for ‘design’
   (b) call interior loop
   (c) while (\textit{ifail} < 0) goto (5)
6. If (\textit{ifail} == 0) probably found an optimum, saved in design/objective.

6 References

