Nelder-Mead User's Manual – The Nelder-Mead Method –

Michaël BAUDIN

Version 0.3
September 2009

Contents

Notations

n	number of variables
$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$	the unknown
$\mathbf{x}_0 \in \mathbb{R}^n$	the initial guess
$\mathbf{v} \in \mathbb{R}^n$	a vertex
$S = \{\mathbf{v}_i\}_{i=1,m}$	a complex, where $m \geq n+1$ is the number of vertices
$S = \{\mathbf{v}_i\}_{i=1,n+1}$	a simplex (with $n+1$ vertices)
$(\mathbf{v}_i)_j$	the j-th component of the i-th vertex
S_0	the initial simplex
S_k	the simplex at iteration k
$\mathbf{v}_i^{(k)}$	the vertex i at iteration k
$f_i^k = f\left(\mathbf{v}_i^(k)\right)$	the function value of the vertex i at iteration k
$f : \mathbb{R}^n \to \mathbb{R}$	the cost function

Fig. 1 : Notations used in this document

Chapter 1

Nelder-Mead method

In this chapter, we present Nelder and Mead's [12] algorithm. We begin by the analysis of the algorithm, which is based on a variable shape simplex. Then, we present geometric situations where the various steps of the algorithm are used. In the third part, we present the rate of convergence toward the optimum of the Nelder-Mead algorithm. This part is mainly based on Han and Neumann's paper [4], which makes use of a class of quadratic functions with a special initial simplex. The core of this chapter is the analysis of several numerical experiments which have been performed with the neldermead component. We analyze the behavior of the algorithm on quadratic functions and present several counter examples where the Nelder-Mead algorithm is known to fail.

1.1 Introduction

In this section, we present the Nelder-Mead algorithm for unconstrained optimization. This algorithm is based on the iterative update of a simplex. Then we present various geometric situations which might occur during the algorithm.

1.1.1 Overview

The goal of the Nelder and Mead algorithm is to solve the following unconstrained optimization problem

$$\min f(\mathbf{x}) \tag{1.1}$$

where $\mathbf{x} \in \mathbb{R}^n$, n is the number of optimization parameters and f is the objective function $f : \mathbb{R}^n \to \mathbb{R}$.

The Nelder-Mead method is an improvement over the Spendley's et al. method with the goal of allowing the simplex to vary in *shape*, and not only in *size*, as in Spendley's et al. algorithm.

This algorithms is based on the iterative update of a *simplex* made of $n+1$ points $S = \{\mathbf{v}_i\}_{i=1,n+1}$. Each point in the simplex is called a *vertex* and is associated with a function value $f_i = f(\mathbf{v}_i)$ for $i = 1, n+1$.

The vertices are sorted by increasing function values so that the *best* vertex has index 1 and the *worst* vertex has index $n+1$

$$f_1 \leq f_2 \leq \ldots \leq f_n \leq f_{n+1}. \tag{1.2}$$

The $\mathbf{v}_1$ vertex (resp. the $\mathbf{v}_{n+1}$ vertex) is called the *best* vertex (resp. *worst*), because it is associated with the lowest (resp. highest) function value.

The centroid of the simplex $\overline{\mathbf{x}}(j)$ is the center of the vertices where the vertex $\mathbf{v}_j$ has been excluded. This centroid is

$$\overline{\mathbf{x}}(j) = \frac{1}{n} \sum_{i=1,n+1,i\neq j} \mathbf{v}_i. \tag{1.3}$$

The algorithm makes use of one coefficient $\rho > 0$, called the reflection factor. The standard value of this coefficient is $\rho = 1$. The algorithm attempts to replace some vertex $\mathbf{v}_j$ by a new vertex $\mathbf{x}(\rho, j)$ on the line from the vertex $\mathbf{v}_j$ to the centroid $\overline{\mathbf{x}}(j)$. The new vertex $\mathbf{x}(\rho, j)$ is defined by

$$\mathbf{x}(\rho, j) = (1+\rho)\overline{\mathbf{x}}(j) - \rho \mathbf{v}_j. \tag{1.4}$$

1.1.2 Algorithm

In this section, we analyze the Nelder-Mead algorithm, which is presented in figure 1.1.

The Nelder-Mead algorithm makes use of four parameters: the coefficient of reflection ρ, expansion χ, contraction γ and shrinkage σ. When the expansion or contraction steps are performed, the shape of the simplex is changed, thus "adapting itself to the local landscape" [12].

These parameters should satisfy the following inequalities [12, 8]

$$\rho > 0, \qquad \chi > 1, \qquad \chi > \rho, \qquad 0 < \gamma < 1 \qquad \text{and} \qquad 0 < \sigma < 1. \tag{1.5}$$

The standard values for these coefficients are

$$\rho = 1, \qquad \chi = 2, \qquad \gamma = \frac{1}{2} \qquad \text{and} \qquad \sigma = \frac{1}{2}. \tag{1.6}$$

In [7], the Nelder-Mead algorithm is presented with other parameter names, that is $\mu_r = \rho$, $\mu_e = \rho\chi$, $\mu_{ic} = -\gamma$ and $\mu_{oc} = \rho\gamma$. These coefficients must satisfy the following inequality

$$-1 < \mu_{ic} < 0 < \mu_{oc} < \mu_r < \mu_e. \tag{1.7}$$

At each iteration, we compute the centroid $\overline{\mathbf{x}}(n+1)$ where the worst vertex $\mathbf{v}_{n+1}$ has been excluded. This centroid is

$$\overline{\mathbf{x}}(n+1) = \frac{1}{n} \sum_{i=1,n} \mathbf{v}_i. \tag{1.8}$$

```
Compute an initial simplex $S_0$
Sorts the vertices $S_0$ with increasing function values
$S \leftarrow S_0$
while $\sigma(S) > tol$ do
  $\overline{x} \leftarrow \overline{x}(n+1)$
  $x_r \leftarrow x(\rho, n+1)$ {Reflect}
  $f_r \leftarrow f(x_r)$
  if $f_r < f_1$ then
    $x_e \leftarrow x(\rho\chi, n+1)$ {Expand}
    $f_e \leftarrow f(x_e)$
    if $f_e < f_r$ then
      Accept $x_e$
    else
      Accept $x_r$
    end if
  else if $f_1 \leq f_r < f_n$ then
    Accept $x_r$
  else if $f_n \leq f_r < f_{n+1}$ then
    $x_c \leftarrow x(\rho\gamma, n+1)$ {Outside contraction}
    $f_c \leftarrow f(x_c)$
    if $f_c < f_r$ then
      Accept $x_c$
    else
      Compute the points $x_i = x_1 + \sigma(x_i - x_1)$, $i = 2, n+1$ {Shrink}
      Compute $f_i = f(\mathbf{v}_i)$ for $i = 2, n+1$
    end if
  else
    $x_c \leftarrow x(-\gamma, n+1)$ {Inside contraction}
    $f_c \leftarrow f(x_c)$
    if $f_c < f_{n+1}$ then
      Accept $x_c$
    else
      Compute the points $x_i = x_1 + \sigma(x_i - x_1)$, $i = 2, n+1$ {Shrink}
      Compute $f_i = f(\mathbf{v}_i)$ for $i = 2, n+1$
    end if
  end if
  Sort the vertices of $S$ with increasing function values
end while
```

Fig. 1.1 : Nelder-Mead algorithm – Standard version

We perform a reflection with respect to the worst vertex $\mathbf{v}_{n+1}$, which creates the reflected point $\mathbf{x}_r$ defined by

$$\mathbf{x}_r = \mathbf{x}(\rho, n+1) = (1+\rho)\overline{\mathbf{x}}(n+1) - \rho \mathbf{v}_{n+1} \tag{1.9}$$

We then compute the function value of the reflected point as $f_r = f(\mathbf{x}_r)$.

From that point, there are several possibilities, which are listed below. Most steps try to replace the worst vertex $\mathbf{v}_{n+1}$ by a better point, which is computed depending on the context.

- In the case where $f_r < f_1$, the reflected point $\mathbf{x}_r$ were able to improve (i.e. reduce) the function value. In that case, the algorithm tries to expand the simplex so that the function value is improved even more. The expansion point is computed by

$$\mathbf{x}_e = \mathbf{x}(\rho\chi, n+1) = (1+\rho\chi)\overline{\mathbf{x}}(n+1) - \rho\chi \mathbf{v}_{n+1} \tag{1.10}$$

  and the function is computed at this point, i.e. we compute $f_e = f(\mathbf{x}_e)$. If the expansion point allows to improve the function value, the worst vertex $\mathbf{v}_{n+1}$ is rejected from the simplex and the expansion point $\mathbf{x}_e$ is accepted. If not, the reflection point $\mathbf{x}_r$ is accepted.

- In the case where $f_1 \leq f_r < f_n$, the worst vertex $\mathbf{v}_{n+1}$ is rejected from the simplex and the reflected point $\mathbf{x}_r$ is accepted.

- In the case where $f_n \leq f_r < f_{n+1}$, we consider the point

$$\mathbf{x}_c = \mathbf{x}(\rho\gamma, n+1) = (1+\rho\gamma)\overline{\mathbf{x}}(n+1) - \rho\gamma \mathbf{v}_{n+1} \tag{1.11}$$

  is considered. If the point $\mathbf{x}_c$ is better than the reflection point $\mathbf{x}_r$, then it is accepted. If not, a shrink step is performed, where all vertices are moved toward the best vertex $\mathbf{v}_1$.

- In other cases, we consider the point

$$\mathbf{x}_c = \mathbf{x}(-\gamma, n+1) = (1-\gamma)\overline{\mathbf{x}}(n+1) + \gamma \mathbf{v}_{n+1}. \tag{1.12}$$

  If the point $\mathbf{x}_c$ is better than the worst vertex $\mathbf{x}_{n+1}$, then it is accepted. If not, a shrink step is performed.

The algorithm from figure 1.1 is the most popular variant of the Nelder-Mead algorithm. But the original paper is based on a "greedy" expansion, where the expansion point is accepted if it is better than the best point (and not if it is better than the reflection point). This "greedy" version is implemented in AS47 by O'Neill in [13] and the corresponding algorithm is presented in figure 1.2.

```
[...]

  $\mathbf{x}_e \leftarrow \mathbf{x}(\rho\chi, n+1)$ {Expand}
  $f_e \leftarrow f(\mathbf{x}_e)$
  if $f_e < f_1$ then
    Accept $\mathbf{x}_e$
  else
    Accept $\mathbf{x}_r$
  end if

[...]
```

Fig. 1.2 : Nelder-Mead algorithm – Greedy version

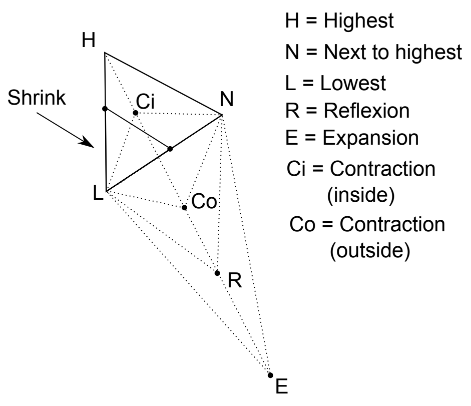

Fig. 1.3 : Nelder-Mead simplex steps

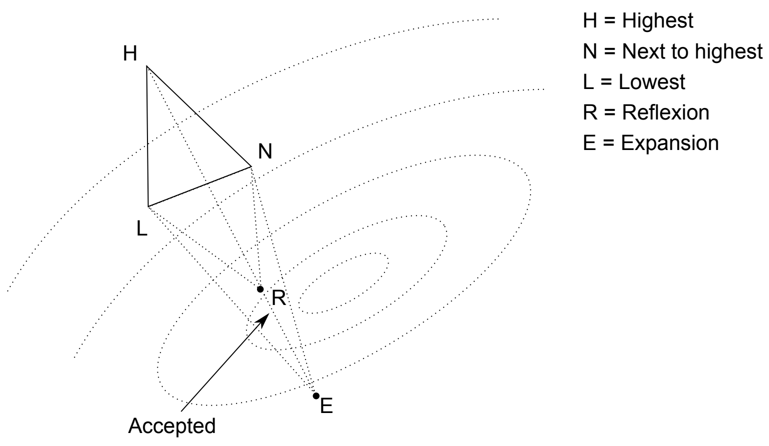

Fig. 1.4 : Nelder-Mead simplex moves – Reflection

1.2 Geometric analysis

The figure 1.3 presents the various moves of the simplex in the Nelder-Mead algorithm.

The figures 1.4 to 1.9 present the detailed situations when each type of step occur. We emphasize that these figures are not the result of numerical experiments. These figures been created in order to illustrate the following specific points of the algorithm.

- Obviously, the expansion step is performed when the simplex is far away from the optimum. The direction of descent is then followed and the worst vertex is moved into that direction.
- When the reflection step is performed, the simplex is getting close to an valley, since the expansion point does not improve the function value.
- When the simplex is near the optimum, the inside and outside contraction steps may be performed, which allows to decrease the size of the simplex. The figure 1.6, which illustrates the inside contraction step, happens in ”good” situations. As presented in section 1.5.4, applying repeatedly the inside contraction step can transform the simplex into a degenerate simplex, which may let the algorithm converge to a non stationnary point.
- The shrink steps (be it after an outside contraction or an inside contraction) occurs only in very special situations. In practical experiments, shrink steps are rare.

1.3 Automatic restarts

In this section, we describe an algorithm which enables the user to perform automatic restarts when a search has failed. A condition is used to detect that a false minimum has been reached. We describe the automatic restart algorithm as well as the conditions used to detect a false minimum.

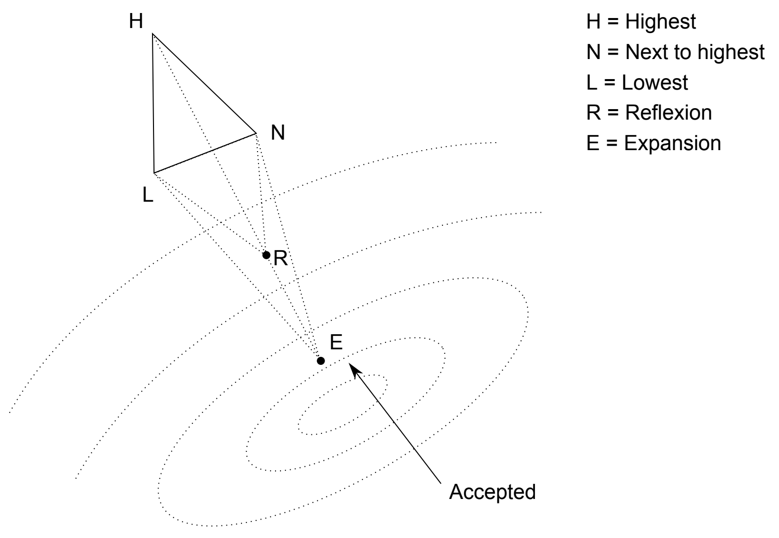

Fig. 1.5 : Nelder-Mead simplex moves – Expansion

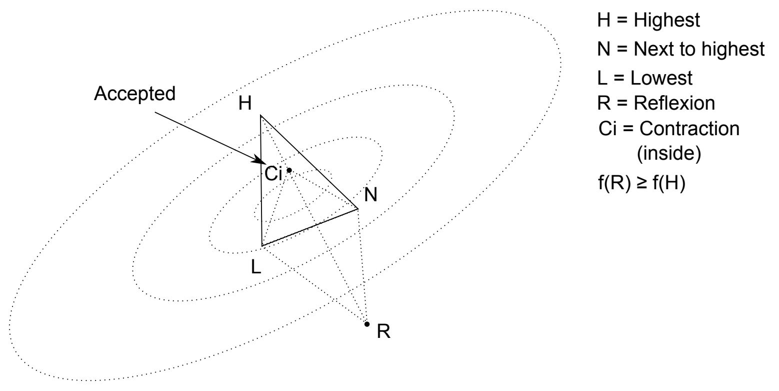

Fig. 1.6 : Nelder-Mead simplex moves - Inside contraction

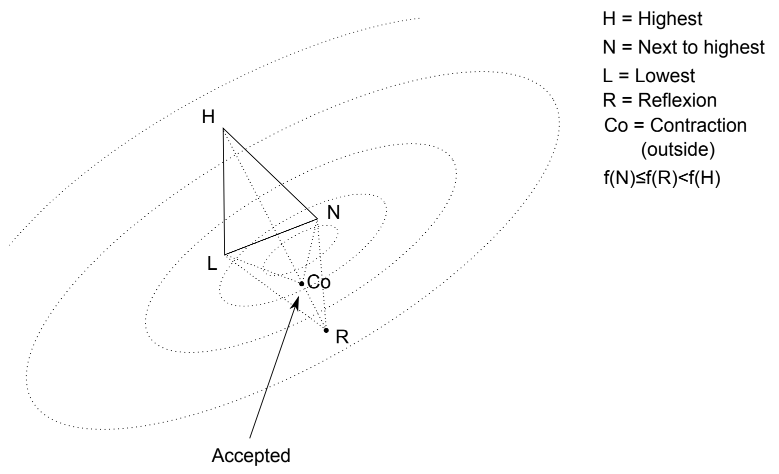

Fig. 1.7 : Nelder-Mead simplex moves – Outside contraction

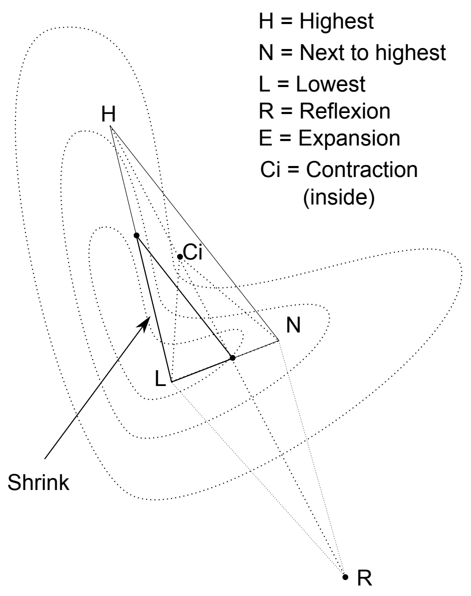

Fig. 1.8 : Nelder-Mead simplex moves – Shrink after inside contraction.

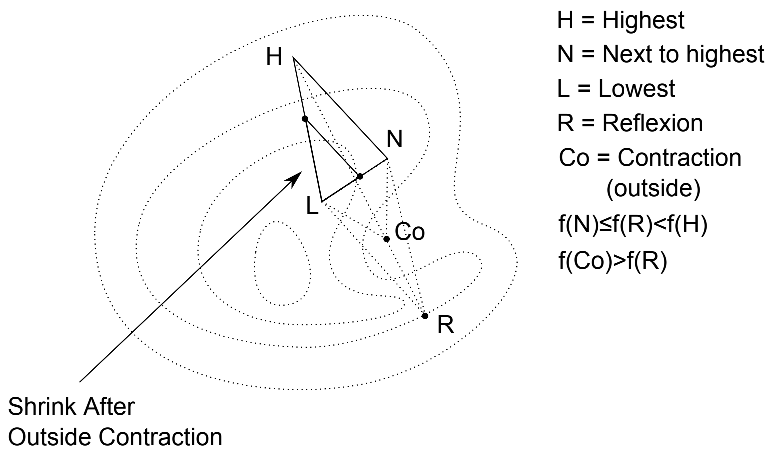

Fig. 1.9 : Nelder-Mead simplex moves – Shrink after outside contraction

1.3.1 Automatic restart algorithm

In this section, we present the automatic restart algorithm.

The goal of this algorithm is to detect that a false minimum has been found, a situation which may occur with the Nelder-Mead algorithm, as we are going to see in the numerical experiments section. These problems are known by practitionners since decades and several authors have tried to detect and solve this specific problem.

In 1971, O'Neill published a fortran 77 implementation of the Nelder-Mead algorithm [13]. In order to check that the algorithm has converged, a factorial test is used. This test will be detailed later in this section. If a false minimum is found by this test, O'Neill suggests to restart the algorithm.

In 1998, Mc Kinnon [9] showed a simple objective function for which the Nelder-Mead algorithm fails to converge to a minimum and, instead, converge to a non-stationnary point. In this numerical experiment, the simplex degenerates toward a single point. In 1999, Kelley [6] shows that restarting the algorithm allows to converge toward the global minimum. In order to detect the convergence problem, Kelley adapted the sufficient decrease condition which is classical in the frameword of gradient-based algorithms. When this condition is met, the algorithm is stopped and a restart should be performed.

Scilab provides an automatic restart algorithm, which allows to detect that a false optimum has been reached and that a new search must be performed. The algorithm is based on a loop where a maximum number of restarts is allowed. The default maximum number of restarts is 3, which means that the maximum number of searches is 4.

After a search has been performed, a condition is computed to know whether a restart must be performed. There are two conditions which are implemented:

- O'Neill factorial test,
- Kelley's stagnation condition.

We will analyze these tests later in this section.

Notice that the automatic restarts are available whatever the simplex algorithm, be it the Nelder-Mead variable shape simplex algorithm, Spendley's et al. fixed shape simplex algorithm or any other algorithm. This is because the automatic restart is a loop programmed above the optimization algorithm.

The automatic restart algorithm is presented in 1.10. Notice that, if a false minimum is detected after the maximum number of restart has been reached, the status is set to ”maxrestart”.

1.3.2 O'Neill factorial test

In this sectin, we present O'Neill's factorial test. This algorithm is given a vector of lengths, stored in the *step* variable. It is also given a small value ϵ, which is an step length relative to the *step* variable. The algorithm is presented in figure 1.11.

```
$restartnb \leftarrow 0$
$reached \leftarrow FALSE$
for $i = 1$ to $restartmax + 1$ do
  $search()$
  $istorestart = istorestart()$
  if $NOT(istorestart)$ then
    $reached \leftarrow TRUE$ {Convergence}
    $BREAK$
  end if
  if $i < restartmax$ then
    $restartnb \leftarrow restartnb + 1$ {A restart is going to be performed}
  end if
end for
if $reached$ then
  printf ( "Convergence reached after
else
  printf ( "Convergence not reached after maximum
  $status \leftarrow$ "$maxrestart$"
end if
```

Fig. 1.10 : Nelder-Mead algorithm – Automatic restart algorithm.

```
$\mathbf{x} \leftarrow \mathbf{x}^\star$
$istorestart = FALSE$
for $i = 1$ to $n$ do
    $\delta = step(i) * \epsilon$
    if $\delta == 0.0$ then
        $\delta = \epsilon$
    end if
    $x(i) = x(i) + \delta$
    $fv = f(x)$
    if $fv < fopt$ then
        $istorestart = TRUE$
        break
    end if
    $x(i) = x(i) - \delta - \delta$
    $fv = f(x)$
    if $fv < fopt$ then
        $istorestart = TRUE$
        break
    end if
    $x(i) = x(i) + \delta$
end for
```

Fig. 1.11 : O'Neill's factorial test

O'Neill's factorial test requires a large number of function evaluations, namely 2^n function evaluations. In O'Neill's implementation, the parameter ϵ is set to the constant value $1.e-3$. In Scilab's implementation, this parameter can be customized, thanks to the *-restarteps* option. Its default value is *%eps*, the machine epsilon. In O'Neill's implementation, the parameter *step* is equal to the vector of length used in order to compute the initial simplex. In Scilab's implementation, the two parameters are different, and the *step* used in the factorial test can be customized with the *-restartstep* option. Its default value is 1.0, which is expanded into a vector with size n.

1.3.3 Kelley's stagnation detection

In this section, we present Kelley's stagnation detection, which is based on the simplex gradient, which definition has been presented in chapter **??**.

C.T. Kelley described in [6] a method to detect stagnation of Nelder-Mead's algorithm. In order to detect the convergence problem, Kelley adapted the sufficient decrease condition which is classical in the frameword of gradient-based algorithms. When this condition is met, the algorithm is stopped and a restart should be performed. We first present the sufficient decrease condition in the context of line search methods. We then present the stagnation condition and a variant of this condition.

Line search and sufficient decrease condition

Before presenting the stagnation criteria suggested by Kelley, it is worthwhile to consider a general gradient-based optimization algorithm and to analyse the way to compute the step length.

Consider an optimization algorithm where the update of the current point $\mathbf{x}^k \in \mathbb{R}^n$ is based on the iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{1.13}$$

where $\mathbf{p}_k \in \mathbb{R}^n$ is the direction and $\alpha_k > 0$ is the step length. Assume that the direction $\mathbf{p}_k$ is given and that α_k is unknown. The problem is to find the minimizer of the one dimensional function Φ defined by the equality

$$\Phi(\alpha) = f(\mathbf{x}_k + \alpha p_k), \tag{1.14}$$

for all $\alpha > 0$.

During the computation of the step length α, there is a tradeoff between reducing sufficiently the function value and not spending too much time in doing so. Line search methods aims at providing an efficient solution for this problem. Several algorithms can be designed in order to find such an optimal α, but all rely on a set of conditions which allows to know when to stop the algorithm. Many line search algorithms are based on the Goldstein-Armijo condition [5, 2], which requires that

$$f(\mathbf{x}_k + \alpha p_k) \leq f(\mathbf{x}_k) + c\alpha \nabla f_k^T \mathbf{p}_k, \tag{1.15}$$

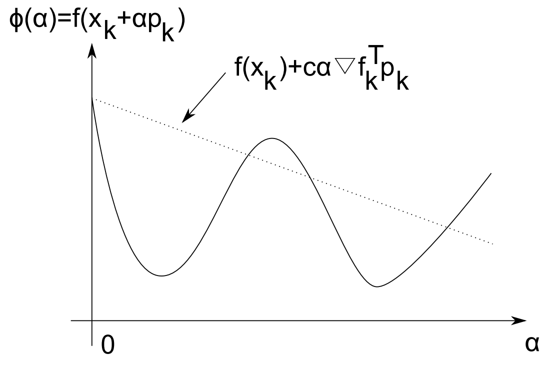

Fig. 1.12 : Sufficient decrease condition

where $c \in (0,1)$ is a given parameter. This condition is presented in figure 1.12. The term $f_k^T \mathbf{p}_k$ is the directionnal derivative of the objective function f along the direction $\mathbf{p}_k$. The Goldstein-Armijo condition ensures that the step length is not too large by requiring that the reduction in f be proportional to the step length α and the directional derivative $f_k^T \mathbf{p}_k$. In practice, the parameter c is often chosen as $c = 10^{-4}$. This implies that the line $f(\mathbf{x}_k) + c\alpha \nabla f_k^T \mathbf{p}_k$ has a slightly decreasing slope, i.e. the condition is rather loose and accept many values of α.

In many line search methods, the Goldstein-Armijo condition is used in combination with another condition, which ensures that the step length α is not too small. This is the additionnal requirement of the Wolfe conditions, also called the curvature condition. We will not detail this further, because the curvature condition is not used in Kelley's stagnation detection criteria.

Stagnation criteria

Let us denote by S_k the simplex at iteration k. We make the assumption that the initial simplex S^0 is nondegenerate, i.e. the condition number of the matrix of simplex directions $\kappa(D(S))$ is finite. We denote by $k \geq 0$ the index of the current iteration. Let us denote by f_1^k the function value at the best vertex $\mathbf{v}_1^{(k)}$, i.e. $f_1^k = f\left(\mathbf{v}_1^{(k)}\right)$.

The derivation is based on the following assumptions.

Assumption 1.3.1 *For all iterations k,*

- *the simplex S_k is nondegenerate,*
- *the vertices are ordered by increasing function value, i.e.*

$$f_1^k \leq f_2^k \leq \ldots \leq f_{n+1}^k, \tag{1.16}$$

- *the best function value is strictly decreasing, i.e. $f_1^{k+1} < f_1^k$.*

If no shrink step occurs in the Nelder-Mead algorithm, then the best function value is indeed decreasing.

Kelley defines a sufficient decrease condition which is analalogous to the sufficient decrease condition for gradient-base algorithms. This condition requires that the $k+1$st iteration satisfies

$$f_1^{k+1} - f_1^k < -c\|\overline{\mathbf{g}}(S_k)\|^2, \tag{1.17}$$

where $\overline{\mathbf{g}}(S_k)$ is the simplex gradient associated with the simplex S_k and $c > 0$ is a small parameter. A typical choice in line-search methods is $c = 10^{-4}$. Kelley suggest in [6] to use 1.17 as a test to detect the stagnation of the Nelder-Mead algorithm.

For consistency, we reproduce below a proposition already presented in chapter **??**.

Proposition 1.3.2 *Let S be a simplex. Let the gradient $\mathbf{g}$ be Lipshitz continuous in a neighbourhood of S with Lipshitz constant L. Consider the euclidian norm $\|.\|$. Then, there is a constant $K > 0$, depending only on L such that*

$$\|\mathbf{g}(\mathbf{v}_1) - \overline{\mathbf{g}}(S)\|_2 \leq K\kappa(S)\sigma_+(S). \tag{1.18}$$

The stagnation detection criteria is based on the following proposition.

Proposition 1.3.3 *Let a sequence of simplices $\{S_k\}_{k\geq 0}$ satisfy assumption 1.3.1. Assume that the sequence $\{f_1^k\}_{k\geq 0}$ is bounded from below. Let the gradient $\mathbf{g}$ of the objective function be Lipshitz continuous in a neighbourhood of $\{S_k\}_{k\geq 0}$ with Lipshitz constant L. Assume that the constant K_k, defined in proposition 1.3.2 is bounded. Assume that the sufficient decrease condition 1.17 is satisfied and that the simplices are so that*

$$\lim_{k\to\infty} \kappa(S_k)\sigma_+(S_k) = 0. \tag{1.19}$$

Therefore, if the best vertex in the simplices converges towards $\mathbf{v}_1^\star$, then $\mathbf{g}(\mathbf{v}_1^\star) = 0$.

Essentially, the proposition states that the condition 1.17 is necessary to get the convergence of the algorithm towards a stationnary point.

Notice that, since the simplex condition number $\kappa(S_k)$ satisfies $\kappa(S_k) \geq 1$, then the the equality 1.19 implies that the size of the simplices converges towards 0.

Proof We first proove that the sequence of simplex gradients $\{\overline{\mathbf{g}}(S_k)\}_{k\geq 0}$ converges toward 0. Notice that the sufficient decrease condition 1.17 can be written as

$$\|\overline{\mathbf{g}}(S_k)\| < \frac{1}{\sqrt{c}}\sqrt{f_1^k - f_1^{k+1}}, \tag{1.20}$$

where the right hand side is positive, by the assumption 1.3.1. By hypothesis, f is uniformly bounded from below and the sequence $\{f_1^k\}_{k\geq 0}$ is stricly decreasing by assumption 1.3.1. Therefore, the sequence $\{f_1^k\}_{k\geq 0}$ converges, which implies that the the sequence $\{f_1^k - f_1^{k+1}\}_{k\geq 0}$ converges to 0. Hence, the inequality 1.20 implies that the sequence $\{\overline{\mathbf{g}}(S_k)\}_{k\geq 0}$ converges towards 0.

Assume that $\mathbf{v}_1^\star$ is an accumulation point of the best vertex of the simplices. We now proove that $\mathbf{v}_1^\star$ is a critical point of the objective function, i.e. we proove that the sequence $\{\mathbf{g}(\mathbf{v}_1^k)\}_{k\geq 0}$ converges towards 0. Notice that we can write the gradient as the sum

$$\mathbf{g}(\mathbf{v}_1^k) = \big(\mathbf{g}(\mathbf{v}_1^k) - \overline{\mathbf{g}}(S_k)\big) + \overline{\mathbf{g}}(S_k), \tag{1.21}$$

which implies

$$\|\mathbf{g}(\mathbf{v}_1^k)\| \leq \|\mathbf{g}(\mathbf{v}_1^k) - \overline{\mathbf{g}}(S_k)\| + \|\overline{\mathbf{g}}(S_k)\|. \tag{1.22}$$

By proposition 1.3.2, there is a constant $K_k > 0$, depending on L and k, such that

$$\|\mathbf{g}(\mathbf{v}_1^k) - \overline{\mathbf{g}}(S_k)\|_2 \leq K_k \kappa(S_k)\sigma_+(S_k). \tag{1.23}$$

By hypothesis, the sequence $\{K_k\}_{k\geq 0}$ is bounded, so that there exists a $K > 0$ so that the inequality $K_k \leq K$, which implies

$$\|\mathbf{g}(\mathbf{v}_1^k) - \overline{\mathbf{g}}(S_k)\|_2 \leq K \kappa(S_k)\sigma_+(S_k). \tag{1.24}$$

We plug the previous inequality into 1.22 and get

$$\|\mathbf{g}(\mathbf{v}_1^k)\| \leq K \kappa(S_k)\sigma_+(S_k) + \|\overline{\mathbf{g}}(S_k)\|. \tag{1.25}$$

We have already prooved that the sequence $\{\overline{\mathbf{g}}(S_k)\}_{k\geq 0}$ converges towards 0. Moreover, by hypothesis, the sequence $\{\kappa(S_k)\sigma_+(S_k)\}_{k\geq 0}$ converges towards 0. Therefore, we have

$$\lim_{k\rightarrow\infty} \mathbf{g}(\mathbf{v}_1^k) = 0, \tag{1.26}$$

which concludes the proof. ∎

Kelley also states a similar theorem which involves noisy functions. These functions are of the form

$$f(\mathbf{x}) = \tilde{f}(\mathbf{x}) + \phi(\mathbf{x}), \tag{1.27}$$

where $\tilde{f}$ is smooth and ϕ is a bounded low-amplitude perturbation. The result is that, if the noise function ϕ has a magnitude smaller than $\sigma_+(S)$, then the proposition 1.3.3 still holds.

A variant of the stagnation criteria

In his book [7], C.T. Kelley suggest a slightly different form for the stagnation criteria 1.17. This variant is based on the fact that the Armijo-Goldstein condition

$$f(\mathbf{x}_k + \alpha p_k) \leq f(\mathbf{x}_k) + c\alpha \nabla f_k^T \mathbf{p}_k, \tag{1.28}$$

distinguish the parameter $c = 10^{-4}$ and the step length $\alpha_k > 0$. In the simplex algorithm, there is no such step length, so that the step length α must be incorporated into the parameter c, which leads to the condition

$$f_1^{k+1} - f_1^k < -c\|\overline{\mathbf{g}}(S_k)\|^2, \tag{1.29}$$

with $c = 10^{-4}$. Now, at the first iteration, the simplex diameter $\sigma_+(S_0)$ might be much smaller that the simplex gradient $\|\overline{\mathbf{g}}(S_k)\|$ so that the previous condition may fail. Kelley address this problem by modifying the previous condition into

$$f_1^{k+1} - f_1^k < -c\frac{\sigma_+(S_0)}{\|\overline{\mathbf{g}}(S_0)\|}\|\overline{\mathbf{g}}(S_k)\|^2. \tag{1.30}$$

1.4 Convergence properties on a quadratic

In this section, we reproduce one result presented by Han and Neumann [4], which states the rate of convergence toward the optimum on a class of quadratic functions with a special initial simplex. Some additional results are also presented in the Phd thesis by Lixing Han [3]. We study a generalized quadratic and use a particular initial simplex. We show that the vertices follow a recurrence equation, which is associated with a characteristic equation. The study of the roots of these characteristic equations give an insight of the behavior of the Nelder-Mead algorithm when the dimension n increases.

Let us suppose than we want to minimize the function

$$f(\mathbf{x}) = x_1^2 + \ldots + x_n^2 \tag{1.31}$$

with the initial simplex

$$S_0 = \left[\mathbf{0}, \mathbf{v}_1^{(0)}, \ldots, \mathbf{v}_n^{(0)}\right] \tag{1.32}$$

With this choice of the initial simplex, the best vertex remains fixed at $\mathbf{0} = (0, 0, \ldots, 0)^T \in \mathbb{R}^n$. As the cost function 1.31 is strictly convex, the Nelder-Mead method never performs the *shrink* step. Therefore, at each iteration, a new simplex is formed by replacing the worst vertex $\mathbf{v}_n^{(k)}$, by a new, better vertex. Assume that the Nelder-Mead method generates a sequence of simplices $\{S_k\}_{k\geq 0}$ in $\mathbb{R}^n$, where

$$S_k = \left[\mathbf{0}, \mathbf{v}_1^{(k)}, \ldots, \mathbf{v}_n^{(n)}\right] \tag{1.33}$$

We wish that the sequence of simplices $S_k \to \mathbf{0} \in \mathbb{R}^n$ as $k \to \infty$. To measure the progress of convergence, Han and Neumann use the oriented length $\sigma_+(S_k)$ of the simplex S_k, defined by

$$\sigma_+(S) = \max_{i=2,m} \|\mathbf{v}_i - \mathbf{v}_1\|_2. \tag{1.34}$$

We say that a sequence of simplices $\{S_k\}_{k\geq 0}$ converges to the minimizer $\mathbf{0} \in \mathbb{R}^n$ of the function in equation 1.31 if $\lim_{k\to\infty} \sigma_+(S_k) = 0$.

We measure the rate of convergence defined by

$$\rho(S_0, n) = \lim \sup_{k\to\infty} \left(\sum_{i=0,k-1} \frac{\sigma(S_{i+1})}{\sigma(S_i)} \right)^{1/k}. \tag{1.35}$$

That definition can be viewed as the geometric mean of the ratio of the oriented lengths between successive simplices and the minimizer 0. This definition implies

$$\rho(S_0, n) = \lim \sup_{k\to\infty} \left(\frac{\sigma(S_{k+1})}{\sigma(S_0)} \right)^{1/k}. \tag{1.36}$$

According to the definition, the algorithm is convergent if $\rho(S_0, n) < 1$. The larger the $\rho(S_0, n)$, the slower the convergence. In particular, the convergence is very slow when $\rho(S_0, n)$ is close to 1. The analysis is based on the fact that the Nelder-Mead method generates a sequence of simplices in $\mathbb{R}^n$ satisfying

$$S_k = \left[\mathbf{0}, \mathbf{v}^{(k+n-1)}, \ldots, \mathbf{v}^{(k+1)}, \mathbf{v}^{(k)}\right], \tag{1.37}$$

where $\mathbf{0}, \mathbf{v}^{(k+n-1)}, \ldots, \mathbf{v}^{(k+1)}, \mathbf{v}^{(k)} \in \mathbb{R}^n$ are the vertices of the $k-th$ simplex, with

$$f(\mathbf{0}) < f\left(\mathbf{v}^{(k+n-1)}\right) < f\left(\mathbf{v}^{(k+1)}\right) < f\left(\mathbf{v}^{(k)}\right), \tag{1.38}$$

for $k \geq 0$.

To simplify the analysis, we consider that only one type of step of the Nelder-Mead method is applied repeatedly. This allows to establish recurrence equations for the successive simplex vertices. As the shrink step is never used, and the expansion steps is never used neither (since the best vertex is already at 0), the analysis focuses on the outside contraction, inside contraction and reflection steps.

The centroid of the n best vertices of S_k is given by

$$\begin{aligned}
\overline{\mathbf{v}}^{(k)} &= \frac{1}{n}\left(\mathbf{v}^{(k+1)} + \ldots + \mathbf{v}^{(k+n-1)} + \mathbf{0}\right) && (1.39)\\
&= \frac{1}{n}\left(\mathbf{v}^{(k+1)} + \ldots + \mathbf{v}^{(k+n-1)}\right) && (1.40)\\
&= \frac{1}{n}\sum_{i=1,n-1} \mathbf{v}^{(k+i)} && (1.41)
\end{aligned}$$

1.4.1 With default parameters

In this section, we analyze the roots of the characteristic equation with *fixed*, standard inside and outside contraction coefficients.

Outside contraction

If the outside contraction step is repeatedly performed with $\mu_{oc} = \rho\gamma = \frac{1}{2}$, then

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} + \frac{1}{2}\left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right). \tag{1.42}$$

By plugging the definition of the centroid 1.41 into the previous equality, we find the recurrence formula

$$2n\mathbf{v}^{(k+n)} - 3\mathbf{v}^{(k+1)} - \ldots - 3\mathbf{v}^{(k+n-1)} + n\mathbf{v}^{(k)} = 0. \tag{1.43}$$

The associated characteristic equation is

$$2n\mu^n - 3\mu^{n-1} - \ldots - 3\mu + n = 0. \tag{1.44}$$

Inside contraction

If the inside contraction step is repeatedly performed with $\mu_{ic} = -\gamma = -\frac{1}{2}$, then

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} - \frac{1}{2}\left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right). \tag{1.45}$$

By plugging the definition of the centroid 1.41 into the previous equality, we find the recurrence formula

$$2n\mathbf{v}^{(k+n)} - \mathbf{v}^{(k+1)} - \ldots - \mathbf{v}^{(k+n-1)} - n\mathbf{v}^{(k)} = 0. \tag{1.46}$$

The associated characteristic equation is

$$2n\mu^n - \mu^{n-1} - \ldots - \mu - n = 0. \tag{1.47}$$

Reflection

If the reflection step is repeatedly performed with $\mu_r = \rho = 1$, then

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} + \left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right). \tag{1.48}$$

By plugging the definition of the centroid 1.41 into the previous equality, we find the recurrence formula

$$n\mathbf{v}^{(k+n)} - 2\mathbf{v}^{(k+1)} - \ldots - 2\mathbf{v}^{(k+n-1)} + n\mathbf{v}^{(k)} = 0. \tag{1.49}$$

The associated characteristic equation is

$$n\mu^n - 2\mu^{n-1} - \ldots - 2\mu + n = 0. \tag{1.50}$$

The recurrence equations 1.44, 1.47 and 1.50 are linear. Their general solutions are of the form

$$\mathbf{v}^{(k)} = \mu_1^k\mathbf{a}_1 + \ldots + \mu_n^k\mathbf{a}_n, \tag{1.51}$$

where $\{\mu_i\}_{i=1,n}$ are the roots of the characteristic equations and $\{\mathbf{a}_i\}_{i=1,n} \in \mathbb{C}^n$ are independent vectors such that $\mathbf{v}^{(k)} \in \mathbb{R}^n$ for all $k \geq 0$.

The analysis by Han and Neumann [4] gives a deep understanding of the convergence rate for this particular situation. For $n = 1$, they show that the convergence rate is $\frac{1}{2}$. For $n = 2$, the convergence rate is $\frac{\sqrt{2}}{2} \approx 0.7$ with a particular choice for the initial simplex. For $n \geq 3$, Han and Neumann [4] perform a numerical analysis of the roots.

In the following Scilab script, we compute the roots of these 3 characteristic equations.

```
//
// computeroots1 --
//   Compute the roots of the characteristic equations of
//   usual Nelder-Mead method.
//
function computeroots1 ( n )
  // Polynomial for outside contraction :
  // n - 3x - ... - 3x^(n-1) + 2n x^(n) = 0
  mprintf("Polynomial␣for␣outside␣contraction␣:\n");
  coeffs = zeros(1,n+1);
  coeffs(1) = n
  coeffs(2:n) = -3
  coeffs(n+1) = 2 * n
  p=poly(coeffs,"x","coeff")
  disp(p)
  mprintf("Roots␣:\n");
  r = roots(p)
  for i=1:n
    mprintf("Root␣#%d/%d␣|%s|=%f\n", i, length(r),string(r(i)),abs(r(i)))
  end
  // Polynomial for inside contraction :
  // - n - x - ... - x^(n-1) + 2n x^(n)= 0
  mprintf("Polynomial␣for␣inside␣contraction␣:\n");
  coeffs = zeros(1,n+1);
  coeffs(1) = -n
  coeffs(2:n) = -1
  coeffs(n+1) = 2 * n
  p=poly(coeffs,"x","coeff")
  disp(p)
  mprintf("Roots␣:\n");
  r = roots(p)
  for i=1:n
    mprintf("Root␣#%d/%d␣|%s|=%f\n", i, length(r),string(r(i)),abs(r(i)))
  end
  // Polynomial for reflection :
  // n - 2x - ... - 2x^(n-1) + n x^(n) = 0
  mprintf("Polynomial␣for␣reflection␣:\n");
  coeffs = zeros(1,n+1);
  coeffs(1) = n
  coeffs(2:n) = -2
  coeffs(n+1) = n
  p=poly(coeffs,"x","coeff")
  disp(p)
  r = roots(p)
  mprintf("Roots␣:\n");
  for i=1:n
    mprintf("Root␣#%d/%d␣|%s|=%f\n", i, length(r),string(r(i)),abs(r(i)))
  end
endfunction
```

If we execute the previous script with $n = 10$, the following output is produced.

```
-->computeroots1 ( 10 )
Polynomial for outside contraction :

                    2     3     4     5     6     7     8      9     10
     10 - 3x - 3x - 3x - 3x - 3x - 3x - 3x - 3x - 3x + 20x
Roots :
Root #1/10 |0.5822700+%i*0.7362568|=0.938676
Root #2/10 |0.5822700-%i*0.7362568|=0.938676
Root #3/10 |-0.5439060+%i*0.7651230|=0.938747
Root #4/10 |-0.5439060-%i*0.7651230|=0.938747
Root #5/10 |0.9093766+%i*0.0471756|=0.910599
Root #6/10 |0.9093766-%i*0.0471756|=0.910599
Root #7/10 |0.0191306+%i*0.9385387|=0.938734
Root #8/10 |0.0191306-%i*0.9385387|=0.938734
Root #9/10 |-0.8918713+%i*0.2929516|=0.938752
Root #10/10 |-0.8918713-%i*0.2929516|=0.938752
Polynomial for inside contraction :

              2   3   4   5   6   7   8    9    10
  - 10 - x - x - x - x - x - x - x - x - x + 20x
Roots :
Root #1/10 |0.7461586+%i*0.5514088|=0.927795
Root #2/10 |0.7461586-%i*0.5514088|=0.927795
Root #3/10 |-0.2879931+%i*0.8802612|=0.926175
Root #4/10 |-0.2879931-%i*0.8802612|=0.926175
Root #5/10 |-0.9260704|=0.926070
Root #6/10 |0.9933286|=0.993329
Root #7/10 |0.2829249+%i*0.8821821|=0.926440
Root #8/10 |0.2829249-%i*0.8821821|=0.926440
Root #9/10 |-0.7497195+%i*0.5436596|=0.926091
Root #10/10 |-0.7497195-%i*0.5436596|=0.926091
Polynomial for reflection :

                2    3    4    5    6    7    8     9    10
    10 - 2x - 2x - 2x - 2x - 2x - 2x - 2x - 2x - 2x + 10x
Roots :
Root #1/10 |0.6172695+%i*0.7867517|=1.000000
Root #2/10 |0.6172695-%i*0.7867517|=1.000000
Root #3/10 |-0.5801834+%i*0.8144859|=1.000000
Root #4/10 |-0.5801834-%i*0.8144859|=1.000000
Root #5/10 |0.9946011+%i*0.1037722|=1.000000
Root #6/10 |0.9946011-%i*0.1037722|=1.000000
Root #7/10 |0.0184670+%i*0.9998295|=1.000000
Root #8/10 |0.0184670-%i*0.9998295|=1.000000
Root #9/10 |-0.9501543+%i*0.3117800|=1.000000
Root #10/10 |-0.9501543-%i*0.3117800|=1.000000
```

The following Scilab script allows to compute the minimum and the maximum of the modulus of the roots. The ”e” option of the ”roots” command has been used to force the use of the eigenvalues of the companion matrix as the computational method. The default algorithm, based on the Jenkins-Traub Rpoly method is generating a convergence error and cannot be used in this case.

```
function [rminoc , rmaxoc , rminic , rmaxic] = computeroots1_abstract ( n )
  // Polynomial for outside contraction :
  // n - 3x - ... - 3x^(n-1) + 2n x^(n) = 0
  coeffs = zeros(1,n+1);
  coeffs(1) = n
  coeffs(2:n) = -3
  coeffs(n+1) = 2 * n
  p=poly(coeffs,"x","coeff")
  r = roots(p , "e")
  rminoc = min(abs(r))
  rmaxoc = max(abs(r))
  // Polynomial for inside contraction :
  // - n - x - ... - x^(n-1) + 2n x^(n)= 0
  coeffs = zeros(1,n+1);
  coeffs(1) = -n
  coeffs(2:n) = -1
  coeffs(n+1) = 2 * n
  p=poly(coeffs,"x","coeff")
  r = roots(p , "e")
  rminic = min(abs(r))
  rmaxic = max(abs(r))
  mprintf("%d␣&␣%f␣&␣%f␣&␣%f␣&␣%f\\\\\n", n, rminoc, rmaxoc, rminic, rmaxic)
endfunction

function drawfigure1 ( nbmax )
  rminoctable = zeros(1,nbmax)
  rmaxoctable = zeros(1,nbmax)
  rminictable = zeros(1,nbmax)
  rmaxictable = zeros(1,nbmax)
  for n = 1 : nbmax
    [rminoc , rmaxoc , rminic , rmaxic] = computeroots1_abstract ( n )
    rminoctable ( n ) = rminoc
    rmaxoctable ( n ) = rmaxoc
    rminictable ( n ) = rminic
    rmaxictable ( n ) = rmaxic
  end
  plot2d ( 1:nbmax , [ rminoctable' , rmaxoctable' , rminictable' , rmaxictable' ] )
  f = gcf();
  f.children.title.text = "Nelder-Mead␣characteristic␣equation␣roots";
  f.children.x_label.text = "Number␣of␣variables␣(n)";
  f.children.y_label.text = "Roots␣of␣the␣characteristic␣equation";
  captions(f.children.children.children,["R-max-IC","R-min-IC","R-max-OC","R-min-OC"]);
  f.children.children(1).legend_location="in_lower_right";
  for i = 1:4
  mypoly = f.children.children(2).children(i);
  mypoly.foreground=i;
  mypoly.line_style=i;
  end
  xs2png(0,"neldermead-roots.png");
endfunction
```

For the reflection characteristic equation, the roots all have a unity modulus. The minimum and maximum roots of the inside contraction ("ic" in the table) and outside contraction ("oc" in the table) steps are presented in table 1.13. These roots are presented graphically in figure 1.14. We see that the roots start from 0.5 when $n = 1$ and converge rapidly toward 1 when $n \to \infty$.

1.4.2 With variable parameters

In this section, we analyze the roots of the characteristic equation with *variable* inside and outside contraction coefficients.

Outside contraction

If the outside contraction step is repeatedly performed with variable $\mu_{oc} \in [0, \mu_r[$, then

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} + \mu_{oc}\left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right) \tag{1.52}$$

n	$\min_{i=1,n} \mu_i^{oc}$	$\max_{i=1,n} \mu_i^{oc}$	$\min_{i=1,n} \mu_i^{ic}$	$\max_{i=1,n} \mu_i^{ic}$
1	0.500000	0.500000	0.500000	0.500000
2	0.707107	0.707107	0.593070	0.843070
3	0.776392	0.829484	0.734210	0.927534
4	0.817185	0.865296	0.802877	0.958740
5	0.844788	0.888347	0.845192	0.973459
6	0.864910	0.904300	0.872620	0.981522
7	0.880302	0.916187	0.892043	0.986406
8	0.892487	0.925383	0.906346	0.989584
9	0.902388	0.932736	0.917365	0.991766
10	0.910599	0.938752	0.926070	0.993329
11	0.917524	0.943771	0.933138	0.994485
12	0.923446	0.948022	0.938975	0.995366
13	0.917250	0.951672	0.943883	0.996051
14	0.912414	0.954840	0.948062	0.996595
15	0.912203	0.962451	0.951666	0.997034
16	0.913435	0.968356	0.954803	0.997393
17	0.915298	0.972835	0.957559	0.997691
18	0.917450	0.976361	0.959999	0.997940
19	0.919720	0.979207	0.962175	0.998151
20	0.922013	0.981547	0.964127	0.998331
21	0.924279	0.983500	0.965888	0.998487
22	0.926487	0.985150	0.967484	0.998621
23	0.928621	0.986559	0.968938	0.998738
24	0.930674	0.987773	0.970268	0.998841
25	0.932640	0.988826	0.971488	0.998932
26	0.934520	0.989747	0.972613	0.999013
27	0.936316	0.990557	0.973652	0.999085
28	0.938030	0.991274	0.974616	0.999149
29	0.939666	0.991911	0.975511	0.999207
30	0.941226	0.992480	0.976346	0.999259
31	0.942715	0.992991	0.977126	0.999306
32	0.944137	0.993451	0.977856	0.999348
33	0.945495	0.993867	0.978540	0.999387
34	0.946793	0.994244	0.979184	0.999423
35	0.948034	0.994587	0.979791	0.999455
36	0.949222	0.994900	0.980363	0.999485
37	0.950359	0.995187	0.980903	0.999513
38	0.951449	0.995450	0.981415	0.999538
39	0.952494	0.995692	0.981900	0.999561
40	0.953496	0.995915	0.982360	0.999583
45	0.957952	0.996807	0.984350	0.999671
50	0.961645	0.997435	0.985937	0.999733
55	0.964752	0.997894	0.987232	0.999779
60	0.967399	0.998240	0.988308	0.999815
65	0.969679	0.998507	0.989217	0.999842
70	0.971665	0.998718	0.989995	0.999864
75	0.973407	0.998887	0.990669	0.999881
80	0.974949	0.999024	0.991257	0.999896
85	0.976323	0.999138	0.991776	0.999908
90	0.977555	0.999233	0.992236	0.999918
95	0.978665	0.999313	0.992648	0.999926
100	0.979671	0.999381	0.993018	0.999933

Fig. 1.13 : Roots of the characteristic equations of the Nelder-Mead method with standard coefficients. (Some results are not displayed to make the table fit the page).

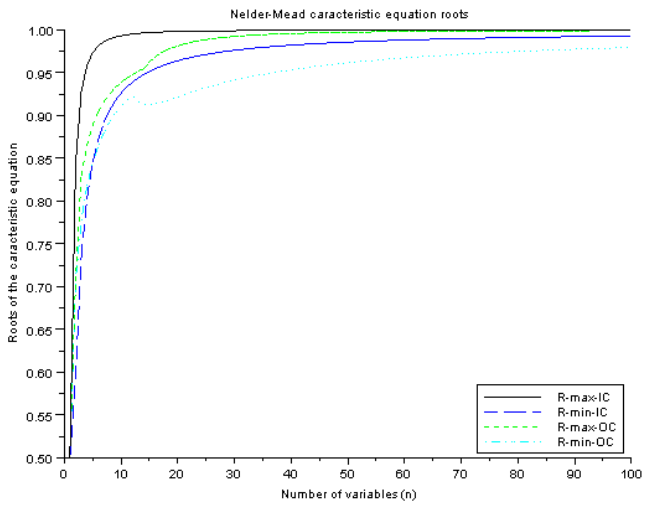

Fig. 1.14 : Modulus of the roots of the characteristic equations of the Nelder-Mead method with standard coefficients – R-max-IC is the maximum of the modulus of the root of the Inside Contraction steps

$$= \quad (1+\mu_{oc})\overline{\mathbf{v}}^{(k)} - \mu_{oc}\mathbf{v}^{(k)} \tag{1.53}$$

By plugging the definition of the centroid into the previous equality, we find the recurrence formula

$$n\mathbf{v}^{(k+n)} - (1+\mu_{oc})\mathbf{v}^{(k+1)} - \ldots - (1+\mu_{oc})\mathbf{v}^{(k+n-1)} + n\mu_{oc}\mathbf{v}^{(k)} = 0 \tag{1.54}$$

The associated characteristic equation is

$$n\mu^n - (1+\mu_{oc})\mu^{n-1} - \ldots - (1+\mu_{oc})\mu + n\mu_{oc} = 0. \tag{1.55}$$

Inside contraction

We suppose that the inside contraction step is repeatedly performed with $-1 < \mu_{ic} < 0$. The characteristic equation is the same as 1.55, but it is here studied in the range $\mu_{ic} \in]-1, 0[$.

To study the convergence of the method, we simply have to study the roots of equation 1.55, where the range $]-1, 0[$ corresponds to the inside contraction (with $-1/2$ as the standard value) and where the range $]0, \mu_r[$ corresponds to the outside contraction (with $1/2$ as the standard value).

In the following Scilab script, we compute the minimum and maximum root of the characteristic equation, with n fixed.

```
//
// rootsvariable --
//   Compute roots of the characteristic equation
//   of Nelder-Mead with variable coefficient mu.
// Polynomial for outside/inside contraction :
// n mu - (1+mu)x - ... - (1+mu)x^(n-1) + n x^(n) = 0
//
function [rmin , rmax] = rootsvariable ( n , mu )
```

```
  coeffs = zeros(1,n+1);
  coeffs(1) = n * mu
  coeffs(2:n) = -(1+mu)
  coeffs(n+1) = n
  p=poly(coeffs,"x","coeff")
  r = roots(p , "e")
  rmin = min(abs(r))
  rmax = max(abs(r))
  mprintf("%f␣&␣%f␣&␣%f\\\\\n", mu, rmin, rmax)
endfunction

function drawfigure_variable ( n , nmumax )
  rmintable = zeros(1,nmumax)
  rmaxtable = zeros(1,nmumax)
  mutable = linspace ( -1 , 1 , nmumax )
  for index = 1 : nmumax
    mu = mutable ( index )
    [rmin , rmax ] = rootsvariable ( n , mu )
    rmintable ( index ) = rmin
    rmaxtable ( index ) = rmax
  end
  plot2d ( mutable , [ rmintable' , rmaxtable' ] )
  f = gcf();
  pause
  f.children.title.text = "Nelder-Mead␣characteristic␣equation␣roots";
  f.children.x_label.text = "Contraction␣coefficient";
  f.children.y_label.text = "Roots␣of␣the␣characteristic␣equation";
  captions(f.children.children.children,["R-max","R-min"]);
  f.children.children(1).legend_location="in_lower_right";
  for i = 1:2
  mypoly = f.children.children(2).children(i);
  mypoly.foreground=i;
  mypoly.line_style=i;
  end
  xs2png(0,"neldermead-roots-variable.png");
endfunction
```

The figure 1.15 presents the minimum and maximum modulus of the roots of the characteristic equation with $n = 10$. The result is that when μ_{oc} is close to 0, the minimum root has a modulus close to 0. The maximum root remains close to 1, whatever the value of the contraction coefficient. This result would mean that either modifying the contraction coefficient has no effect (because the maximum modulus of the roots is close to 1) or diminishing the contraction coefficient should improve the convergence speed (because the minimum modulus of the roots gets closer to 0). This is the expected result because the more the contraction coefficient is close to 0, the more the new vertex is close to 0, which is, in our particular situation, the global minimizer. No general conclusion can be drawn from this single experiment.

1.5 Numerical experiments

In this section, we present some numerical experiments with the Nelder-Mead algorithm. The two first numerical experiments involve simple quadratic functions. These experiments allows to see the difference between Spendley's et al. algorithm and the Nelder-Mead algorithm. We then present several experiments taken from the bibliography. The O'Neill experiments [13] are performed in order to check that our algorithm is a correct implementation. We then present several numerical experiments where the Nelder-Mead does not converge properly. We analyze the Mc Kinnon counter example from [9]. We show the behavior of the Nelder-Mead simplex method for a family of examples which cause the method to converge to a non stationnary point. We analyze the counter examples presented by Han in his Phd thesis [3]. In these experiments,

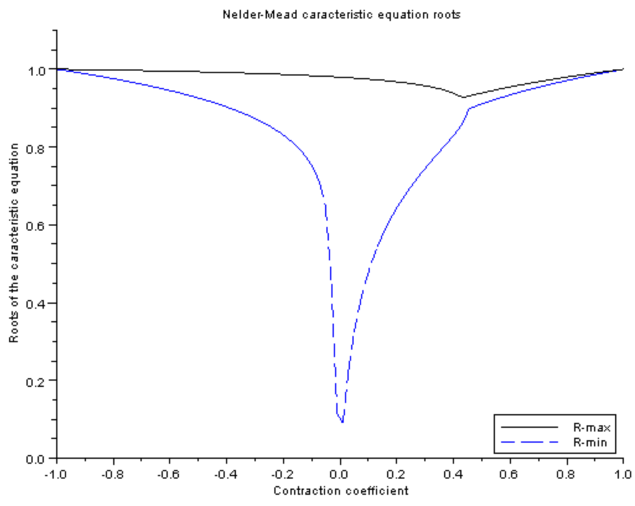

Fig. 1.15 : Modulus of the roots of the characteristic equations of the Nelder-Mead method with variable contraction coefficient and $n = 10$ – R-max is the maximum of the modulus of the root of the characteristic equation

the Nelder-Mead algorithm degenerates by applying repeatedly the inside contraction step. We also reproduce numerical experiments extracted from Torczon's Phd Thesis [16], where Virginia Torczon presents the multi-directional direct search algorithm.

1.5.1 Quadratic function

The function we try to minimize is the following quadratic in 2 dimensions

$$f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2. \tag{1.56}$$

The stopping criteria is based on the relative size of the simplex with respect to the size of the initial simplex

$$\sigma_+(S) < tol \times \sigma_+(S_0), \tag{1.57}$$

where the tolerance is set to $tol = 10^{-8}$.

The initial simplex is a regular simplex with unit length.

The following Scilab script allows to perform the optimization.

```
function [ y , index ] = quadratic ( x , index )
  y = x(1)^2 + x(2)^2 - x(1) * x(2);
endfunction
nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",quadratic);
nm = neldermead_configure(nm,"-x0",[2.0 2.0]');
nm = neldermead_configure(nm,"-maxiter",100);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolxmethod",%f);
nm = neldermead_configure(nm,"-tolsimplexizerelative",1.e-8);
nm = neldermead_configure(nm,"-simplex0method","spendley");
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_search(nm);
neldermead_display(nm);
nm = neldermead_destroy(nm);
```

The numerical results are presented in table 1.16.

Iterations	65
Function Evaluations	130
x_0	$(2.0, 2.0)$
Relative tolerance on simplex size	10^{-8}
Exact $x^\star$	$(0., 0.)$
Computed $x^\star$	$(-2.519D-09, 7.332D-10)$
Computed $f(x^\star)$	$8.728930e-018$

Fig. 1.16 : Numerical experiment with Nelder-Mead method on the quadratic function $f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2$

The various simplices generated during the iterations are presented in figure 1.17.

The figure 1.18 presents the history of the oriented length of the simplex. The length is updated at each iteration, which generates a continuous evolution of the length, compared to the step-by-step evolution of the simplex with the Spendley et al. algorithm.

The convergence is quite fast in this case, since less than 70 iterations allow to get a function value lower than 10^{-15}, as shown in figure 1.19.

Badly scaled quadratic function

The function we try to minimize is the following quadratic in 2 dimensions

$$f(x_1, x_2) = a x_1^2 + x_2^2, \tag{1.58}$$

where $a > 0$ is a chosen scaling parameter. The more a is large, the more difficult the problem is to solve with the simplex algorithm.

We set the maximum number of function evaluations to 400. The initial simplex is a regular simplex with unit length. The stopping criteria is based on the relative size of the simplex with respect to the size of the initial simplex

$$\sigma_+(S) < tol \times \sigma_+(S_0), \tag{1.59}$$

where the tolerance is set to $tol = 10^{-8}$.

The following Scilab script allows to perform the optimization.

```
a = 100.0;
function [ y , index ] = quadratic ( x , index )
  y = a * x(1)^2 + x(2)^2;
endfunction
```

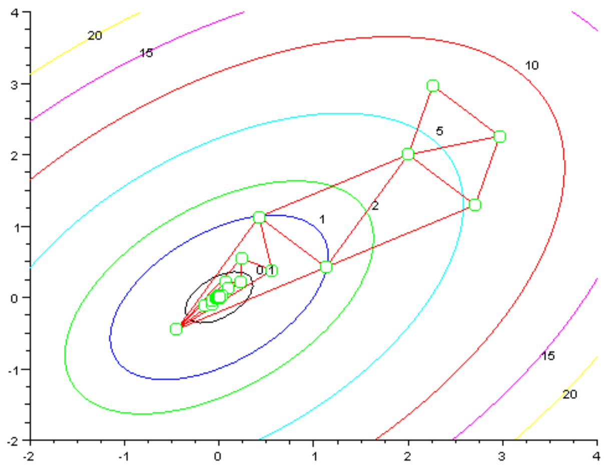

Fig. 1.17 : Nelder-Mead numerical experiment – history of simplex

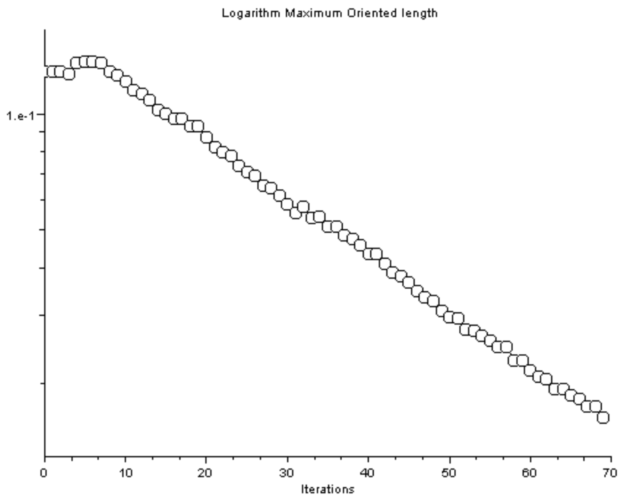

Fig. 1.18 : Nelder-Mead numerical experiment – History of logarithm of length of simplex

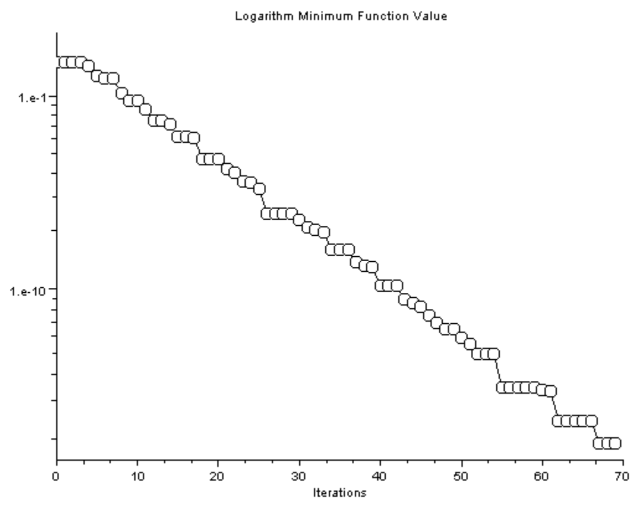

Fig. 1.19 : Nelder-Mead numerical experiment – History of logarithm of function

```
nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",quadratic);
nm = neldermead_configure(nm,"-x0",[10.0 10.0]');
nm = neldermead_configure(nm,"-maxiter",400);
nm = neldermead_configure(nm,"-maxfunevals",400);
nm = neldermead_configure(nm,"-tolxmethod",%f);
nm = neldermead_configure(nm,"-tolsimplexizerelative",1.e-8);
nm = neldermead_configure(nm,"-simplex0method","spendley");
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_search(nm);
neldermead_display(nm);
nm = neldermead_destroy(nm);
```

The numerical results are presented in table 1.20, where the experiment is presented for $a = 100$. We can check that the number of function evaluation (161 function evaluations) is much lower than the number for the fixed shape Spendley et al. method (400 function evaluations) and that the function value at optimum is very accurate ($f(x^\star) \approx 10^{-17}$ compared to Spendley's et al. $f(x^\star) \approx 0.08$).

In figure 1.21, we analyze the behavior of the method with respect to scaling. We check that the method behaves very smoothly, with a very small number of additional function evaluations when the scaling deteriorates. This shows how much the Nelder-Mead algorithms improves over Spendley's et al. method.

1.5.2 Sensitivity to dimension

In this section, we try to reproduce the result presented by Han and Neumann [4], which shows that the convergence rate of the Nelder-Mead algorithms rapidly deteriorates when the number

	Nelder-Mead	Spendley et al.
Iterations	82	340
Function Evaluations	164	Max=400
a	100.0	100.0
x_0	$(10.0, 10.0)$	$(10.0, 10.0)$
Initial simplex	regular	regular
Initial simplex length	1.0	1.0
Relative tolerance on simplex size	10^{-8}	10^{-8}
Exact $x^\star$	$(0., 0.)$	$(0., 0.)$
Computed $x^\star$	$(-2.D-10-1.D-09)$	$(0.001, 0.2)$
Computed $f(x^\star)$	$1.D-017$	0.08

Fig. 1.20 : Numerical experiment with Nelder-Mead method on a badly scaled quadratic function. The variable shape Nelder-Mead algorithm improves the accuracy of the result compared to the fixed shaped Spendley et al. method.

a	Function Evaluations	Computed $f(x^\star)$	Computed $x^\star$
1.0	147	$1.856133e-017$	$(1.920D-09, -3.857D-09)$
10.0	156	$6.299459e-017$	$(2.482D-09, 1.188D-09)$
100.0	164	$1.140383e-017$	$(-2.859D-10, -1.797D-09)$
1000.0	173	$2.189830e-018$	$(-2.356D-12, 1.478D-09)$
10000.0	189	$1.128684e-017$	$(2.409D-11, -2.341D-09)$

Fig. 1.21 : Numerical experiment with Nelder-Mead method on a badly scaled quadratic function

of variables increases. The function we try to minimize is the following quadratic in n-dimensions

$$f(\mathbf{x}) = \sum_{i=1,n} x_i^2. \tag{1.60}$$

The initial simplex is given to the solver. The first vertex is the origin ; this vertex is never updated during the iterations. The other vertices are based on uniform random numbers in the interval $[-1, 1]$. The vertices $i = 2, n+1$ are computed from

$$\mathbf{v}_i^{(0)} = 2rand(n, 1) - 1, \tag{1.61}$$

as prescribed by [4]. In Scilab, the *rand* function returns a matrix of uniform random numbers in the interval $[0, 1)$.

The stopping criteria is based on the absolute size of the simplex, i.e. the simulation is stopped when

$$\sigma_+(S) < tol, \tag{1.62}$$

where the tolerance is set to $tol = 10^{-8}$.

We perform the experiment for $n = 1, \ldots, 19$. For each experiment, we compute the convergence rate from

$$\rho(S_0, n) = \left(\frac{\sigma(S_k)}{\sigma(S_0)}\right)^{1/k}, \tag{1.63}$$

where k is the number of iterations.

The following Scilab script allows to perform the optimization.

```
function [ f , index ] = quadracticn ( x , index )
  f = sum(x.^2);
endfunction
//
// solvepb --
//   Find the solution for the given number of dimensions
//
function [nbfevals , niter , rho] = solvepb ( n )
  rand("seed",0)
  nm = neldermead_new ();
  nm = neldermead_configure(nm,"-numberofvariables",n);
  nm = neldermead_configure(nm,"-function",quadracticn);
  nm = neldermead_configure(nm,"-x0",zeros(n,1));
  nm = neldermead_configure(nm,"-maxiter",2000);
  nm = neldermead_configure(nm,"-maxfunevals",2000);
  nm = neldermead_configure(nm,"-tolxmethod",%f);
  nm = neldermead_configure(nm,"-tolsimplexizerelative",0.0);
  nm = neldermead_configure(nm,"-tolsimplexizeabsolute",1.e-8);
  nm = neldermead_configure(nm,"-simplex0method","given");
  coords (1,1:n) = zeros(1,n);
  for i = 2:n+1
    coords (i,1:n) = 2.0 * rand(1,n) - 1.0;
  end
  nm = neldermead_configure(nm,"-coords0",coords);
  nm = neldermead_configure(nm,"-method","variable");
  nm = neldermead_search(nm);
  si0 = neldermead_get ( nm , "-simplex0" );
  sigma0 = optimsimplex_size ( si0 , "sigmaplus" );
  siopt = neldermead_get ( nm , "-simplexopt" );
  sigmaopt = optimsimplex_size ( siopt , "sigmaplus" );
  niter = neldermead_get ( nm , "-iterations" );
  rho = (sigmaopt/sigma0)^(1.0/niter);
  nbfevals = neldermead_get ( nm , "-funevals" );
  mprintf ( "%d %d %d %f\n", n , nbfevals , niter , rho );
  nm = neldermead_destroy(nm);
endfunction
// Perform the 20 experiments
for n = 1:20
  [nbfevals niter rho] = solvepb ( n );
  array_rho(n) = rho;
  array_nbfevals(n) = nbfevals;
  array_niter(n) = niter;
end
```

The figure 1.22 presents the results of this experiment. The rate of convergence, as measured by $\rho(S_0, n)$ converges rapidly toward 1.

We check that the number of function evaluations increases approximately linearly with the dimension of the problem in figure 1.23. A rough rule of thumb is that, for $n = 1, 19$, the number of function evaluations is equal to $100n$.

The figure 1.24 presents the rate of convergence depending on the number of variables. The figure shows that the rate of convergence rapidly gets close to 1 when the number of variables increases. That shows that the rate of convergence is slower and slower as the number of variables increases, as explained by Han & Neumann.

n	Function evaluations	Iterations	$\rho(S_0, n)$
1	56	27	0.513002
2	113	55	0.712168
3	224	139	0.874043
4	300	187	0.904293
5	388	249	0.927305
6	484	314	0.941782
7	583	383	0.951880
8	657	430	0.956872
9	716	462	0.959721
10	853	565	0.966588
11	910	596	0.968266
12	1033	685	0.972288
13	1025	653	0.970857
14	1216	806	0.976268
15	1303	864	0.977778
16	1399	929	0.979316
17	1440	943	0.979596
18	1730	1193	0.983774
19	1695	1131	0.982881
20	1775	1185	0.983603

Fig. 1.22 : Numerical experiment with Nelder-Mead method on a generalized quadratic function

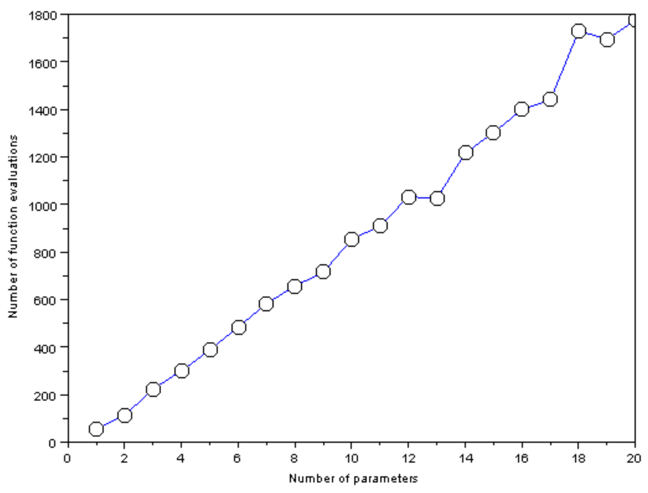

Fig. 1.23 : Nelder-Mead numerical experiment – Number of function evaluations depending on the number of variables

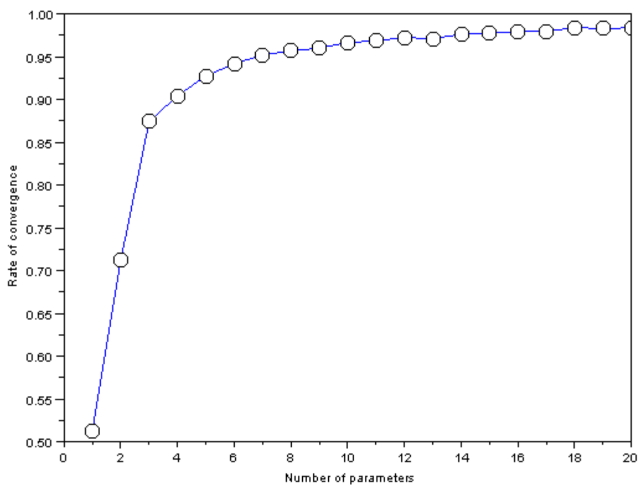

Fig. 1.24 : Nelder-Mead numerical experiment – Rate of convergence depending on the number of variables

1.5.3 O'Neill test cases

In this section, we present the results by O'Neill, who implemented a fortran 77 version of the Nelder-Mead algorithm [13].

The O'Neill implementation of the Nelder-Mead algorithm has the following particularities

- the initial simplex is computed from the axes and a (single) length,
- the stopping rule is based on variance (not standard deviation) of function value,
- the expansion is greedy, i.e. the expansion point is accepted if it is better than the lower point,
- an automatic restart is performed if a factorial test shows that the computed optimum is greater than a local point computed with a relative epsilon equal to 1.e-3 and a step equal to the length of the initial simplex.

The following tests are presented by O'Neill :

- Rosenbrock's parabolic valley [15]

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{1.64}$$

  with starting point $\mathbf{x}_0 = (x_1, x_2) = (-1.2, 1)^T$. The function value at initial guess is $f(\mathbf{x}_0) = 24.2$. The solution is $\mathbf{x}^\star = (1, 1)^T$ where the function value is $f(\mathbf{x}^\star) = 0$.

- Powell's quartic function [14]

$$f(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \tag{1.65}$$

  with starting point $\mathbf{x}_0 = (x_1, x_2, x_3, x_4) = (3, -1, 0, 1)^T$. The function value at initial guess is $f(\mathbf{x}_0) = 215.$. The solution is $\mathbf{x}^\star = (0, 0, 0, 0)^T$ where the function value is $f(\mathbf{x}^\star) = 0.$.

- Fletcher and Powell's helical valley [1]

$$f(x_1, x_2, x_3) = 100\left(x_3 + 10\theta(x_1, x_2)\right)^2 + \left(\sqrt{x_1^2 + x_2^2} - 1\right)^2 + x_3^2 \tag{1.66}$$

  where

$$2\pi\theta(x_1, x_2) = \begin{cases} \arctan(x_2, x_1), & \text{if } x_1 > 0 \\ \pi + \arctan(x_2, x_1), & \text{if } x_1 < 0 \end{cases} \tag{1.67}$$

  with starting point $\mathbf{x}_0 = (x_1, x_2, x_3) = (-1, 0, 0)$. The function value at initial guess is $f(\mathbf{x}_0) = 2500$. The solution is $\mathbf{x}^\star = (1, 0, 0)^T$ where the function value is $f(\mathbf{x}^\star) = 0.$. Note that since arctan(0/0) is not defined neither the function f on the line $(0, 0, x_3)$. This line is excluded by assigning a very large value to the function.

- the sum of powers

$$f(x_1, \ldots, x_{10}) = \sum_{i=1,10} x_i^4 \tag{1.68}$$

  with starting point $\mathbf{x}_0 = (x_1, \ldots, x_{10}) = (1, \ldots, 1)$. The function value at initial guess is $f(\mathbf{x}_0) = 10$. The solution is $\mathbf{x}^\star = (0, \ldots, 0)^T$ where the function value is $f(\mathbf{x}^\star) = 0$..

The parameters are set to (following O'Neill's notations)

- $REQMIN = 10^{-16}$, the absolute tolerance on the variance of the function values in the simplex,
- $STEP = 1.0$, the absolute side length of the initial simplex,
- $ICOUNT = 1000$, the maximum number of function evaluations.

The following Scilab script allows to define the objective functions.

```
// Rosenbrock's "banana" function
// initialguess [-1.2 1.0]
// xoptimum [1.0 1.0}
// foptimum 0.0
function [ y , index ] = rosenbrock ( x , index )
y = 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
// Powell's quartic valley
// initialguess [3.0 -1.0 0.0 1.0]
// xoptimum [0.0 0.0 0.0 0.0]
// foptimum 0.0
function [ f , index ] = powellquartic ( x , index )
  f = (x(1)+10.0*x(2))^2 + 5.0 * (x(3)-x(4))^2 + (x(2)-2.0*x(3))^4 + 10.0 * (x(1) - x(4))^4
endfunction
// Fletcher and Powell helical valley
// initialguess [-1.0 0.0 0.0]
// xoptimum [1.0 0.0 0.0]
// foptimum 0.0
function [ f , index ] = fletcherpowellhelical ( x , index )
  rho = sqrt(x(1) * x(1) + x(2) * x(2))
  twopi = 2 * %pi
  if ( x(1)==0.0 ) then
    f = 1.e154
  else
    if ( x(1)>0 ) then
      theta = atan(x(2)/x(1)) / twopi
    elseif ( x(1)<0 ) then
      theta = (%pi + atan(x(2)/x(1))) / twopi
    end
    f =  100.0 * (x(3)-10.0*theta)^2 + (rho - 1.0)^2 + x(3)*x(3)
  end
endfunction
// Sum of powers
// initialguess ones(10,1)
// xoptimum zeros(10,1)
// foptimum 0.0
function [ f , index ] = sumpowers ( x , index )
  f = sum(x(1:10).^4);
endfunction
```

The following Scilab function solves an optimization problem, given the number of parameters, the cost function and the initial guess.

```
//
// solvepb --
//   Find the solution for the given problem.
// Arguments
//   n : number of variables
//   cfun : cost function
//   x0 : initial guess
//
function [nbfevals , niter , nbrestart , fopt , cputime ] = solvepb ( n , cfun , x0 )
  tic();
  nm = neldermead_new ();
  nm = neldermead_configure(nm,"-numberofvariables",n);
  nm = neldermead_configure(nm,"-function",cfun);
  nm = neldermead_configure(nm,"-x0",x0);
  nm = neldermead_configure(nm,"-maxiter",1000);
  nm = neldermead_configure(nm,"-maxfunevals",1000);
  nm = neldermead_configure(nm,"-tolxmethod",%f);
  nm = neldermead_configure(nm,"-tolsimplexizemethod",%f);
  // Turn ON the tolerance on variance
  nm = neldermead_configure(nm,"-tolvarianceflag",%t);
  nm = neldermead_configure(nm,"-tolabsolutevariance",1.e-16);
  nm = neldermead_configure(nm,"-tolrelativevariance",0.0);
  // Turn ON automatic restart
  nm = neldermead_configure(nm,"-restartflag",%t);
  nm = neldermead_configure(nm,"-restarteps",1.e-3);
  nm = neldermead_configure(nm,"-restartstep",1.0);
  // Turn ON greedy expansion
  nm = neldermead_configure(nm,"-greedy",%t);
  // Set initial simplex to axis-by-axis (this is already the default anyway)
  nm = neldermead_configure(nm,"-simplex0method","axes");
  nm = neldermead_configure(nm,"-simplex0length",1.0);
  nm = neldermead_configure(nm,"-method","variable");
  //nm = neldermead_configure(nm,"-verbose",1);
  //nm = neldermead_configure(nm,"-verbosetermination",1);
  //
  // Perform optimization
  //
  nm = neldermead_search(nm);
  //neldermead_display(nm);
  niter = neldermead_get ( nm , "-iterations" );
  nbfevals = neldermead_get ( nm , "-funevals" );
  fopt = neldermead_get ( nm , "-fopt" );
  xopt = neldermead_get ( nm , "-xopt" );
  nbrestart = neldermead_get ( nm , "-restartnb" );
  status = neldermead_get ( nm , "-status" );
  nm = neldermead_destroy(nm);
  cputime = toc();
  mprintf ( "=================================\n")
  mprintf ( "status = %s\n" , status )
  mprintf ( "xopt = [%s]\n" , strcat(string(xopt)," ") )
  mprintf ( "fopt = %e\n" , fopt )
  mprintf ( "niter = %d\n" , niter )
  mprintf ( "nbfevals = %d\n" , nbfevals )
  mprintf ( "nbrestart = %d\n" , nbrestart )
  mprintf ( "cputime = %f\n" , cputime )
  //mprintf ( "%d %d %e %d %f\n", nbfevals , nbrestart , fopt , niter , cputime );
endfunction
```

The following Scilab script solves the 4 cases.

```
// Solve Rosenbrock's
x0 = [-1.2 1.0].';
[nbfevals , niter , nbrestart , fopt , cputime ] = solvepb ( 2 , rosenbrock , x0 );

// Solve Powell's quartic valley
x0 = [3.0 -1.0 0.0 1.0].';
[nbfevals , niter , nbrestart , fopt , cputime ] = solvepb ( 4 , powellquartic , x0 );

// Solve Fletcher and Powell helical valley
x0 = [-1.0 0.0 0.0].';
[nbfevals , niter , nbrestart , fopt , cputime ] = solvepb ( 3 , fletcherpowellhelical , x0 );

// Solve Sum of powers
x0 = ones(10,1);
[nbfevals , niter , nbrestart , fopt , cputime ] = solvepb ( 10 , sumpowers , x0 );
```

The table 1.25 presents the results which were computed by O'Neill compared with Scilab's. For most experiments, the results are very close in terms of number of function evaluations. The problem #4 exhibits a different behavior than the results presented by O'Neill. For Scilab, the tolerance on variance of function values is reach after 3 restarts, whereas for O'Neill, the algorithm is restarted once and gives the result with 474 function evaluations. We did not find any explanation for this behavior. A possible cause of difference may be the floating point system which are different and may generate different simplices in the algorithms. Although the CPU times cannot be compared (the article is dated 1972 !), let's mention that the numerical experiment were performed by O'Neill on a ICL 4-50 where the two problem 1 and 2 were solved in 3.34 seconds and the problems 3 and 4 were solved in 22.25 seconds.

Author	Problem	Function Evaluations	Number Of Restarts	Function Value	Iterations	CPU Time
O'Neill	1	148	0	3.19e-9	?	?
Scilab	1	155	0	1.158612e-007	80	0.625000
O'Neill	2	209	0	7.35e-8	?	?
Scilab	2	234	0	1.072588e-008	126	0.938000
O'Neill	3	250	0	5.29e-9	?	?
Scilab	3	263	0	4.560288e-008	137	1.037000
O'Neill	4	474	1	3.80e-7	?	?
Scilab	4	616	3	3.370756e-008	402	2.949000

Fig. 1.25 : Numerical experiment with Nelder-Mead method on O'Neill test cases - O'Neill results and Scilab's results

1.5.4 Mc Kinnon: convergence to a non stationnary point

In this section, we analyze the Mc Kinnon counter example from [9]. We show the behavior of the Nelder-Mead simplex method for a family of examples which cause the method to converge to a non stationnary point.

Consider a simplex in two dimensions with vertices at 0 (i.e. the origin), $\mathbf{v}^{(n+1)}$ and $\mathbf{v}^{(n)}$. Assume that

$$f(0) < f(\mathbf{v}^{(n+1)}) < f(\mathbf{v}^{(n)}). \tag{1.69}$$

The centroid of the simplex is $\overline{\mathbf{v}} = \mathbf{v}^{(n+1)}/2$, the midpoint of the line joining the best and second vertex. The reflected point is then computed as

$$\mathbf{r}^{(n)} = \overline{\mathbf{v}} + \rho(\overline{\mathbf{v}} - \mathbf{v}^{(n)}) = \mathbf{v}^{(n+1)} - \mathbf{v}^{(n)} \tag{1.70}$$

Assume that the reflection point $\mathbf{r}^{(n)}$ is rejected, i.e. that $f(\mathbf{v}^{(n)}) < f(\mathbf{r}^{(n)})$. In this case, the inside contraction step is taken and the point $\mathbf{v}^{(n+2)}$ is computed using the reflection factor $-\gamma = -1/2$ so that

$$\mathbf{v}^{(n+2)} = \overline{\mathbf{v}} - \gamma(\overline{\mathbf{v}} - \mathbf{v}^{(n)}) = \frac{1}{4}\mathbf{v}^{(n+1)} - \frac{1}{2}\mathbf{v}^{(n)} \tag{1.71}$$

Assume then that the inside contraction point is accepted, i.e. $f(\mathbf{v}^{(n+2)}) < f(\mathbf{v}^{(n+1)})$. If this sequence of steps repeats, the simplices are subject to the following linear recurrence formula

$$4\mathbf{v}^{(n+2)} - \mathbf{v}^{(n+1)} + 2\mathbf{v}^{(n)} = 0 \tag{1.72}$$

Their general solutions are of the form

$$\mathbf{v}^{(n)} = \lambda_1^k a_1 + \lambda_2^k a_2 \tag{1.73}$$

where $\lambda_{ii=1,2}$ are the roots of the characteristic equation and $a_{ii=1,2} \in \mathbb{R}^n$. The characteristic equation is

$$4\lambda^2 - \lambda + 2\lambda = 0 \tag{1.74}$$

and has the roots

$$\lambda_1 = \frac{1+\sqrt{33}}{8} \approx 0.84307, \qquad \lambda_2 = \frac{1-\sqrt{33}}{8} \approx -0.59307 \tag{1.75}$$

After Mc Kinnon has presented the computation of the roots of the characteristic equation, he presents a special initial simplex for which the simplices degenerates because of repeated failure by inside contraction (RFIC in his article). Consider the initial simplex with vertices $\mathbf{v}^{(0)} = (1,1)$ and $\mathbf{v}^{(1)} = (\lambda_1, \lambda_2)$ and 0. If follows that the particular solution for these initial conditions is $\mathbf{v}^{(n)} = (\lambda_1^n, \lambda_2^n)$.

Consider the function $f(x_1, x_2)$ given by

$$\begin{aligned} f(x_1,x_2) &= \theta\phi|x_1|^\tau + x_2 + x_2^2, \qquad x_1 \leq 0, & (1.76)\\ &= \theta x_1^\tau + x_2 + x_2^2, \qquad x_1 \geq 0. & (1.77)\end{aligned}$$

where θ and ϕ are positive constants. Note that $(0,-1)$ is a descent direction from the origin $(0,0)$ and that f is stricly convex provided $\tau > 1$. f has continuous first derivatives if $\tau > 1$, continuous second derivatives if $\tau > 2$ and continuous third derivatives if $\tau > 3$.

Mc Kinnon computed the conditions on θ, ϕ and τ so that the function values are ordered as expected, i.e. so that the reflection step is rejected and the inside contraction is accepted. Examples of values which makes these equations hold are as follows : for $\tau = 1$, $\theta = 15$ and $\phi = 10$, for $\tau = 2$, $\theta = 6$ and $\phi = 60$ and for $\tau = 3$, $\theta = 6$ and $\phi = 400$.

We consider here the more regular case $\tau = 3$, $\theta = 6$ and $\phi = 400$, i.e. the function is defined by

$$f(x_1,x_2) = \begin{cases} -2400x_1^3 + x_2 + x_2^2, & \text{if } x_1 \leq 0, \\ 6x_1^3 + x_2 + x_2^2, & \text{if } x_1 \geq 0. \end{cases} \tag{1.78}$$

The solution is $\mathbf{x}^\star = (0, -0.5)^T$.

The following Scilab script solves the optimization problem. We must use the "-simplex0method" option so that a user-defined initial simplex is used. Then the "-coords0" allows to define the coordinates of the initial simplex, where each row corresponds to a vertex of the simplex

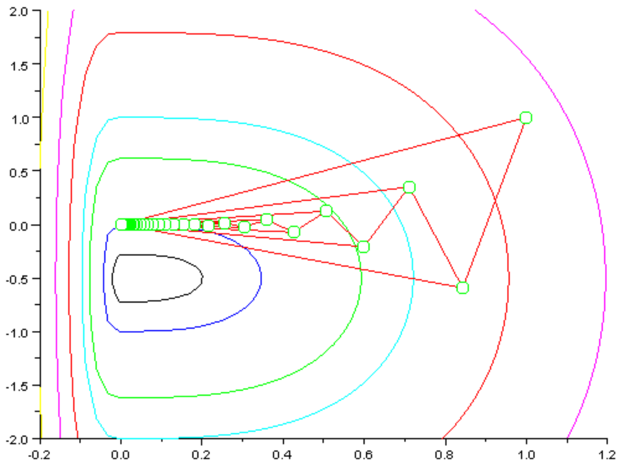

Fig. 1.26 : Nelder-Mead numerical experiment – Mc Kinnon example for convergence toward a non stationnary point

```
function [ f , index ] = mckinnon3 ( x , index )
  if ( length ( x ) ~= 2 )
    error ( 'Error: function expects a two dimensional input\n' );
  end
  tau = 3.0;
  theta = 6.0;
  phi = 400.0;
  if ( x(1) <= 0.0 )
    f = theta * phi * abs ( x(1) ).^tau + x(2) * ( 1.0 + x(2) );
  else
    f = theta       *       x(1).^tau   + x(2) * ( 1.0 + x(2) );
  end
endfunction
lambda1 = (1.0 + sqrt(33.0))/8.0;
lambda2 = (1.0 - sqrt(33.0))/8.0;
coords0 = [
1.0  1.0
0.0  0.0
lambda1 lambda2
];
x0 = [1.0 1.0]';
nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",mckinnon3);
nm = neldermead_configure(nm,"-x0",x0);
nm = neldermead_configure(nm,"-maxiter",200);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolfunrelative",10*%eps);
nm = neldermead_configure(nm,"-tolxrelative",10*%eps);
nm = neldermead_configure(nm,"-simplex0method","given");
nm = neldermead_configure(nm,"-coords0",coords0);
nm = neldermead_search(nm);
neldermead_display(nm);
nm = neldermead_destroy(nm);
```

The figure 1.26 shows the contour plot of this function and the first steps of the Nelder-Mead method. The global minimum is located at $(0, -1/2)$. Notice that the simplex degenerates to the point $(0, 0)$, which is a non stationnary point.

The figure 1.27 presents the first steps of the algorithm in this numerical experiment. Because of the particular shape of the contours of the function, the reflected point is always worse that

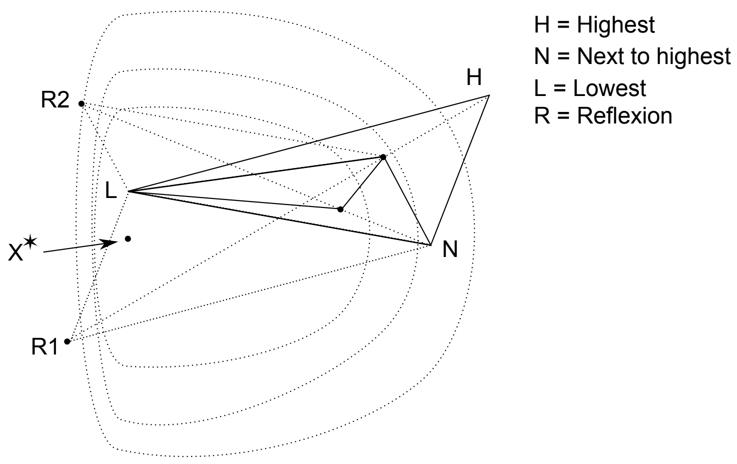

Fig. 1.27 : Nelder-Mead numerical experiment – Detail of the first steps. The simplex converges to a non stationnary point, after repeated inside contractions.

the worst vertex $\mathbf{x}_{n+1}$. This leads to the inside contraction step. The vertices constructed by Mc Kinnon are so that the situation loops without end.

1.5.5 Kelley: oriented restart

Kelley analyzed Mc Kinnon counter example in [7]. He analyzed the evolution of the simplex gradient and found that its norm begins to grow when the simplex start to degenerate. Therefore, Kelley suggest to detect the stagnation of the algorithm by using a termination criteria which is based on a sufficient decrease condition. Once that the stagnation is detected and the algorithm is stopped, restarting the algorithm with a non-degenerated simplex allows to converge toward the global minimum. Kelley advocates the use of the oriented restart, where the new simplex is so that it maximizes the chances of producing a good descent direction at the next iteration.

The following Scilab script solves the optimization problem. We must use the "-simplex0method" option so that a user-defined initial simplex is used. Then the "-coords0" allows to define the coordinates of the initial simplex, where each row corresponds to a vertex of the simplex.

We also use the "-kelleystagnationflag" option, which turns on the termination criteria associated with Kelley's stagnation detection method. Once that the algorithm is stopped, we want to automatically restart the algorithm. This is why we turn on the "-restartflag" option, which enables to perform automatically 3 restarts. After an optimization process, the automatic restart algorithm needs to know if the algorithm must restart or not. By default, the algorithm uses a factorial test, due to O'Neill. This is why we configure the "-restartdetection" to the "kelley" option, which uses Kelley's termination condition as a criteria to determine if a restart must be performed.

```
function [ f , index ] = mckinnon3 ( x , index )
```

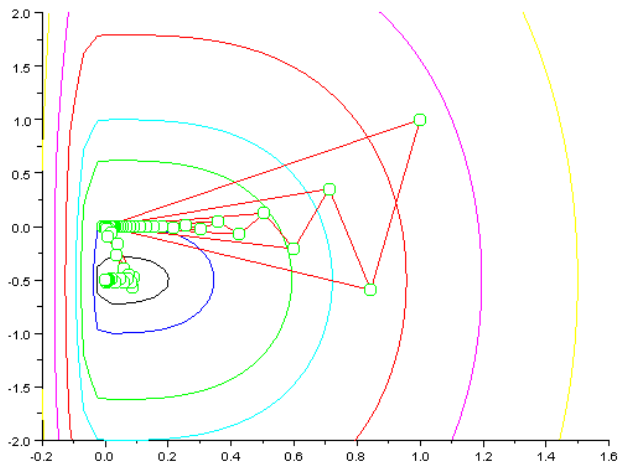

Fig. 1.28 : Nelder-Mead numerical experiment – Mc Kinnon example with Kelley's stagnation detection.

```
  if ( length ( x ) ~= 2 )
    error ( 'Error: function expects a two dimensional input\n' );
  end
  tau = 3.0;
  theta = 6.0;
  phi = 400.0;
  if ( x(1) <= 0.0 )
    f = theta * phi * abs ( x(1) ).^tau + x(2) * ( 1.0 + x(2) );
  else
    f = theta       *       x(1).^tau   + x(2) * ( 1.0 + x(2) );
  end
endfunction
lambda1 = (1.0 + sqrt(33.0))/8.0;
lambda2 = (1.0 - sqrt(33.0))/8.0;
coords0 = [
1.0  1.0
0.0  0.0
lambda1 lambda2
];
x0 = [1.0  1.0]';
nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",mckinnon3);
nm = neldermead_configure(nm,"-x0",x0);
nm = neldermead_configure(nm,"-maxiter",200);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolsimplexizerelative",1.e-6);
nm = neldermead_configure(nm,"-simplex0method","given");
nm = neldermead_configure(nm,"-coords0",coords0);
nm = neldermead_configure(nm,"-kelleystagnationflag",%t);
nm = neldermead_configure(nm,"-restartflag",%t);
nm = neldermead_configure(nm,"-restartdetection","kelley");
nm = neldermead_search(nm);
neldermead_display(nm);
nm = neldermead_destroy(nm);
```

The figure 1.28 presents the first steps of the algorithm in this numerical experiment. We see that the algorithm converges now toward the minimum $\mathbf{x}^\star = (0, -0.5)^T$.

1.5.6 Han counter examples

In his Phd thesis [3], Han presents two counter examples in which the Nelder-Mead algorithm degenerates by applying repeatedly the inside contraction step.

First counter example

The first counter example is based on the function

$$f(x_1, x_2) = x_1^2 + x_2(x_2 + 2)(x_2 - 0.5)(x_2 - 2) \tag{1.79}$$

This function is nonconvex, bounded below and has bounded level sets. The initial simplex is chosen as $S_0 = [(0., -1), (0, 1), (1, 0)]$. Han proves that the Nelder-Mead algorithm generates a sequence of simplices $S_k = [(0., -1), (0, 1), (\frac{1}{2^k}, 0)]$.

```
function [ f , index ] = han1 ( x , index )
  f = x(1)^2 + x(2) * (x(2) + 2.0) * (x(2) - 0.5) * (x(2) - 2.0);
endfunction
coords0 = [
    0.   -1.
    0.    1.
    1.    0.
]
nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",han1);
nm = neldermead_configure(nm,"-x0",[1.0 1.0]');
nm = neldermead_configure(nm,"-maxiter",50);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolfunrelative",10*%eps);
nm = neldermead_configure(nm,"-tolxrelative",10*%eps);
nm = neldermead_configure(nm,"-simplex0method","given");
nm = neldermead_configure(nm,"-coords0",coords0);
nm = neldermead_search(nm);
neldermead_display(nm);
nm = neldermead_destroy(nm);
```

The figure 1.29 presents the isovalues and the simplices during the steps of the Nelder-Mead algorithm. Note that the limit simplex contains no minimizer of the function. The failure is caused by repeated inside contractions.

Second counter example

The second counter example is based on the function

$$f(x_1, x_2) = x_1^2 + \rho(x_2) \tag{1.80}$$

where ρ is a continuous convex function with bounded level sets defined by

$$\begin{cases} \rho(x_2) = 0, & \text{if} \quad |x_2| \leq 1, \\ \rho(x_2) \geq 0, & \text{if} \quad |x_2| > 1. \end{cases} \tag{1.81}$$

The example given by Han for such a ρ function is

$$\rho(x_2) = \begin{cases} 0, & \text{if} \quad |x_2| \leq 1, \\ x_2 - 1, & \text{if} \quad x_2 > 1, \\ -x_2 - 1, & \text{if} \quad x_2 < -1. \end{cases} \tag{1.82}$$

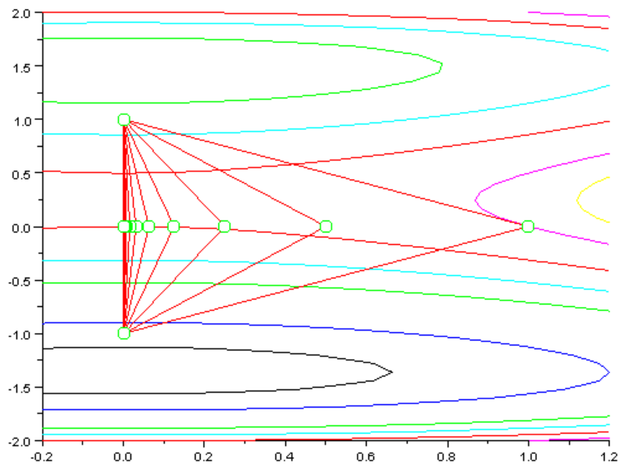

Fig. 1.29 : Nelder-Mead numerical experiment – Han example #1 for convergence toward a non stationnary point

The initial simplex is chosen as $S_0 = [(0., 1/2), (0, -1/2), (1, 0)]$. Han prooves that the Nelder-Mead algorithm generates a sequence of simplices $S_k = [(0., 1/2), (0, -1/2), (\frac{1}{2^k}, 0)]$.

```
function [ f , index ] = han2 ( x , index )
  if abs(x(2)) <= 1.0 then
    rho = 0.0
  elseif x(2) > 1.0 then
    rho = x(2) - 1
  else
    rho = -x(2) - 1
  end
  f = x(1)^2 + rho;
endfunction
coords0 = [
    0.    0.5
    0.   -0.5
    1.    0.
]
nm = neldermead_new ();
nm = neldermead_configure(nm,"-numberofvariables",2);
nm = neldermead_configure(nm,"-function",han2);
nm = neldermead_configure(nm,"-x0",[1.0 1.0]');
nm = neldermead_configure(nm,"-maxiter",50);
nm = neldermead_configure(nm,"-maxfunevals",300);
nm = neldermead_configure(nm,"-tolfunrelative",10*%eps);
nm = neldermead_configure(nm,"-tolxrelative",10*%eps);
nm = neldermead_configure(nm,"-simplex0method","given");
nm = neldermead_configure(nm,"-coords0",coords0);
nm = neldermead_search(nm);
neldermead_display(nm);
nm = neldermead_destroy(nm);
```

The figure 1.30 presents the isovalues and the simplices during the steps of the Nelder-Mead algorithm. The failure is caused by repeated inside contractions.

These two examples of non convergence show that the Nelder-Mead method may unreliable. They also reveal that the Nelder-Mead method can generate simplices which collapse into a degenerate simplex, by applying repeated inside contractions.

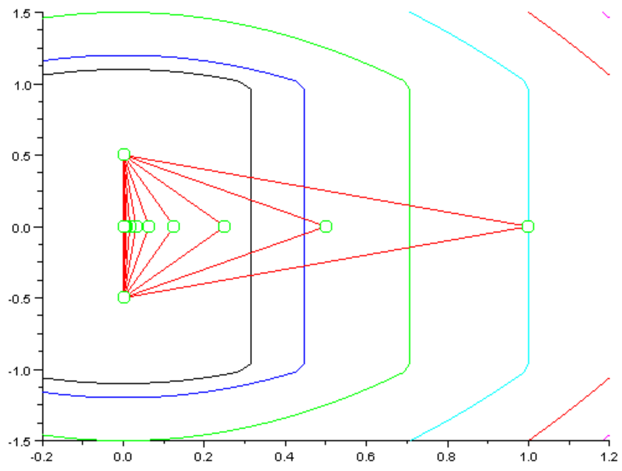

Fig. 1.30 : Nelder-Mead numerical experiment – Han example #2 for convergence toward a non stationnary point

1.5.7 Torczon's numerical experiments

In her Phd Thesis [16], Virginia Torczon presents the multi-directional direct search algorithm. In order to analyze the performances of her new algorithm, she presents some interesting numerical experiments with the Nelder-Mead algorithm. These numerical experiments are based on the collection of test problems [10], published in the ACM by Moré, Garbow and Hillstrom in 1981. These test problems are associated with varying number of variables. In her Phd, Torczon presents numerical experiments with n from 8 to 40. The stopping rule is based on the relative size of the simplex. The angle between the descent direction (given by the worst point and the centroid), and the gradient of the function is computed when the algorithm is stopped. Torczon shows that, when the tolerance on the relative simplex size is decreased, the angle converges toward 90°. This fact is observed even for moderate number of dimensions.

In this section, we try to reproduce Torczon numerical experiments.

All experiments are associated with the following sum of squares cost function

$$f(\mathbf{x}) = \sum_{i=1,m} f_i(\mathbf{x})^2, \tag{1.83}$$

where $m \geq 1$ is the number of functions f_i in the problem.

The stopping criteria is based on the relative size of the simplex and is the following

$$\frac{1}{\Delta} \max_{i=2,n+1} \|\mathbf{v}_i - \mathbf{v}_1\| \leq \epsilon, \tag{1.84}$$

where $\Delta = \max(1, \|\mathbf{v}_1\|)$. Decreasing the value of ϵ allows to get smaller simplex sizes.

The initial simplex is not specified by Virginia Torczon. In our numerical experiments, we choose an axis-by-axis simplex, with an initial length equal to 1.

Penalty #1

The first test function is the *Penalty #1* function :

$$f_i(\mathbf{x}) = 10^{-5/2}(x_i - 1), \qquad i = 1, n \tag{1.85}$$

$$f_{n+1} = -\frac{1}{4} + \sum_{j=1,n} x_j^2. \tag{1.86}$$

The initial guess is given by $\mathbf{x}_0 = ((\mathbf{x}_0)_1, (\mathbf{x}_0)_2, \ldots, (\mathbf{x}_0)_n)^T$ and $(\mathbf{x}_0)_j = j$ for $j = 1, n$.

The problem given by Moré, Garbow and Hillstrom in [10, 11] is associated with the size $n = 4$. The value of the cost function at the initial guess $\mathbf{x}_0 = (1, 2, 3, 4)^T$ is $f(\mathbf{x}_0) = 885.063$. The value of the function at the optimum is given in [10, 11] as $f(\mathbf{x}^\star) = 2.24997d - 5$.

Virginia Torzcon present the results of this numerical experiment with the Penalty #1 test case and $n = 8$. For this particular case, the initial function value is $f(\mathbf{x}_0) = 4.151406.10^4$.

In the following Scilab script, we define the *penalty1* function. We define the function *penalty1_der* which allows to compute the numerical derivative. The use of a global variable is not

```
function [ y , index , n ] = penalty1 ( x , index , n )
  y = 0.0
  for i = 1 : n
    fi = (x(i) - 1) * sqrt(1.e-5)
    y = y + fi^2
  end
  fi = -1/4 + norm(x)^2
  y = y + fi^2
endfunction

function y = penalty1_der ( x , n )
  [ y , index ] = penalty1 ( x , 1 , n )
endfunction
```

The following Scilab function defines the termination criteria, as defined in 1.84.

```
function [ this , terminate , status ] = mystoppingrule ( this , simplex )
  global _DATA_
  v1 = optimsimplex_getx ( simplex , 1 )
  delta = max ( 1.0 , norm(v1) )
  maxnorms = 0.0
  n = neldermead_cget ( this , "-numberofvariables" )
  for i = 2 : n
    vi = optimsimplex_getx ( simplex , i )
    ni = norm ( vi - v1 )
    maxnorms = max ( maxnorms , ni )
  end
  epsilon = _DATA_.epsilon
  if ( maxnorms / delta < epsilon ) then
    terminate = %t
    status = "torczon"
  else
    terminate = %f
  end
endfunction
```

The following *solvepb* function takes as input the dimension of the problem n, the cost function, the initial guess and the tolerance. It uses the *neldermead* component and configures it so that the algorithm uses the specific termination function defined previously.

```
function [nbfevals , niter , fopt , cputime ] = solvepb ( n , cfun , x0 , tolerance )
  tic();
  global _DATA_;
  _DATA_ = tlist ( [
    "T_TORCZON"
    "epsilon"
  ]);
  _DATA_.epsilon = tolerance;

  nm = neldermead_new ();
  nm = neldermead_configure(nm,"-numberofvariables",n);
  nm = neldermead_configure(nm,"-function",cfun);
  nm = neldermead_configure(nm,"-costfargument",n);
  nm = neldermead_configure(nm,"-x0",x0);
  nm = neldermead_configure(nm,"-maxiter",10000);
  nm = neldermead_configure(nm,"-maxfunevals",10000);
  nm = neldermead_configure(nm,"-tolxmethod",%f);
  nm = neldermead_configure(nm,"-tolsimplexizemethod",%f);
  // Turn ON my own termination criteria
  nm = neldermead_configure(nm,"-myterminate",mystoppingrule);
  nm = neldermead_configure(nm,"-myterminateflag",%t);
  //
  // Perform optimization
  //
  nm = neldermead_search(nm);
  niter = neldermead_get ( nm , "-iterations" );
  nbfevals = neldermead_get ( nm , "-funevals" );
  fopt = neldermead_get ( nm , "-fopt" );
  xopt = neldermead_get ( nm , "-xopt" );
  status = neldermead_get ( nm , "-status" );
  nm = neldermead_destroy(nm);
  cputime = toc();
  // Compute angle between gradient and simplex direction
  sopt = neldermead_get ( nm , "-simplexopt" )
  xhigh = optimsimplex_getx ( sopt , n + 1 )
  xbar = optimsimplex_xbar ( sopt )
  d = xbar - xhigh;
  g = derivative ( list ( penalty1_der , n ) , xopt , order=4 );
  cost = -g*d.' / norm(g) / norm(d)
  theta =acosd(cost)
  // Compute condition of matrix of directions
  D = optimsimplex_dirmat ( sopt )
  k = cond ( D )
  // Display result
  mprintf ( "==============================\n")
  mprintf ( "status␣=␣%s\n" , status )
  mprintf ( "Tolerance=%e\n" , tolerance )
  mprintf ( "xopt␣=␣[%s]\n" , strcat(string(xopt),"␣") )
  mprintf ( "fopt␣=␣%e\n" , fopt )
  mprintf ( "niter␣=␣%d\n" , niter )
  mprintf ( "nbfevals␣=␣%d\n" , nbfevals )
  mprintf ( "theta␣=␣%25.15f␣(deg)\n" , theta )
  mprintf ( "cputime␣=␣%f␣(s)\n" , cputime )
  mprintf ( "cond(D)␣=␣%e␣(s)\n" , k )
endfunction
```

We are now able to make a loop, and get the optimum function value for various values of the tolerance use in the termination criteria.

```
x0 = [1 2 3 4 5 6 7 8].';
for tol = [1.e-1 1.e-2 1.e-3 1.e-4 1.e-5 1.e-6 1.e-7]
  [nbfevals , niter , fopt , cputime ] = solvepb ( 8 , penalty1 , x0 , tol );
end
```

The figure 1.31 presents the results of these experiments. As Virginia Torczon, we get an increasing number of function evaluations, with very little progress with respect to the function value. We also get a search direction which becomes increasingly perpendicular to the gradient.

The number of function evaluations is not the same between Torczon's and Scilab so that we can conclude that the algorithm may be different variants of the Nelder-Mead algorithm or uses a different initial simplex. We were not able to explain why the number of function evaluations is so different.

Author	Step Tolerance	$f(\mathbf{v}_1^\star)$	Function Evaluations	Angle (°)
Torzcon	1.e-1	7.0355e-5	1605	89.396677792198
Scilab	1.e-1	9.567114e-5	314	101.297069897149110
Torzcon	1.e-2	6.2912e-5	3360	89.935373548613
Scilab	1.e-2	8.247686e-5	501	88.936037514983468
Torzcon	1.e-3	6.2912e-5	3600	89.994626919197
Scilab	1.e-3	7.485404e-5	1874	90.134605846897529
Torzcon	1.e-4	6.2912e-5	3670	89.999288284747
Scilab	1.e-4	7.481546e-5	2137	90.000107262503008
Torzcon	1.e-5	6.2912e-5	3750	89.999931862232
Scilab	1.e-5	7.481546e-5	2193	90.000366248870506
Torzcon	1.e-6	6.2912e-5	3872	89.999995767877
Scilab	1.e-6	7.427204e-5	4792	90.000006745652769
Torzcon	1.e-7	6.2912e-5	3919	89.999999335010
Scilab	1.e-7	7.427204e-5	4851	89.999996903432063

Fig. 1.31 : Numerical experiment with Nelder-Mead method on penalty #1 test case - Torczon results and Scilab's results

The figure 1.32 presents the condition number of the matrix of simplex direction. When this condition number is high, the simplex is distorted. The numerical experiment shows that the condition number is fastly increasing. This corresponds to the fact that the simplex is increasingly distorted and might explains why the algorithm fails to make any progress.

1.6 Conclusion

The main advantage of the Nelder-Mead algorithm over Spendley et al. algorithm is that the shape of the simplex is dynamically updated. That allows to get a reasonably fast convergence rate on badly scaled quadratics, or more generally when the cost function is made of a sharp valley. Still, the behavior of the algorithm when the dimension of the problem increases is disappointing: the more there are variables, the more the algorithm is slow. In general, it is expected that the number of function evaluations is roughly equal to $100n$, where n is the number of parameters. When the algorithm comes close to the optimum, the simplex becomes more and more distorted, so that less and less progress is made with respect to the value of the cost function. This can measured by the fact that the direction of search becomes more and more perpendicular to the gradient of the cost function. It can also be measure by an increasing value of the condition number of the matrix of simplex directions. Therefore, the user should not require a high accuracy from the algorithm. Nevertheless, in most cases, the Nelder-Mead algorithms provides a good *improvement*

Tolerance	$cond(D)$
1.e-1	1.573141e+001
1.e-2	4.243385e+002
1.e-3	7.375247e+008
1.e-4	1.456121e+009
1.e-5	2.128402e+009
1.e-6	2.323514e+011
1.e-7	3.193495e+010

Fig. 1.32 : Numerical experiment with Nelder-Mead method on penalty #1 test case - Condition number of the matrix of simplex directions

of the solution. In some situations, the simplex can become so distorted that it converges toward a non-stationnary point. In this case, restarting the algorithm with a new nondegenerate simplex allows to converge toward the optimum.

Bibliography

[1] R. Fletcher and M. J. D. Powell. A Rapidly Convergent Descent Method for Minimization. *The Computer Journal*, 6(2):163–168, 1963.

[2] P. E. Gill, W. Murray, and M. H. Wright. *Practical optimization*. Academic Press, London, 1981.

[3] Lixing Han. *Algorithms in Unconstrained Optimization*. Ph.D., The University of Connecticut, 2000.

[4] Lixing Han and Michael Neumann. Effect of dimensionality on the nelder-mead simplex method. *Optimization Methods and Software*, 21(1):1–16, 2006.

[5] Stephen J. Wright Jorge Nocedal. *Numerical Optimization*. Springer, 1999.

[6] C. T. Kelley. Detection and remediation of stagnation in the nelder–mead algorithm using a sufficient decrease condition. *SIAM J. on Optimization*, 10(1):43–55, 1999.

[7] C. T. Kelley. *Iterative Methods for Optimization*, volume 19. SIAM Frontiers in Applied Mathematics, 1999.

[8] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998.

[9] K. I. M. McKinnon. Convergence of the nelder–mead simplex method to a nonstationary point. *SIAM J. on Optimization*, 9(1):148–158, 1998.

[10] J. J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. Algorithm 566: Fortran subroutines for testing unconstrained optimization software [c5], [e4]. *ACM Trans. Math. Softw.*, 7(1):136–140, 1981.

[11] Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. Testing unconstrained optimization software. *ACM Trans. Math. Softw.*, 7(1):17–41, 1981.

[12] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[13] R. O'Neill. Algorithm AS47 - Function minimization using a simplex procedure. *Applied Statistics*, 20(3):338–346, 1971.

[14] M. J. D. Powell. An Iterative Method for Finding Stationary Values of a Function of Several Variables. *The Computer Journal*, 5(2):147–151, 1962.

[15] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, March 1960.

[16] Virginia Joanne Torczon. Multi-directional search: A direct search algorithm for parallel machines. Technical report, Rice University, 1989.

Index