Nelder-Mead User's Manual

Michaël BAUDIN

Abstract

In this document, we present the Nelder-Mead component provided in Scilab. The introduction gives a brief overview of the optimization features of the component and present an introductory example. Then we present some theory associated with the simplex, a geometric concept which is central in the Nelder-Mead algorithm. We present several method to compute an initial simplex. Then we present Spendley's et al. fixed shape unconstrained optimization algorithm. Several numerical experiments are provided, which shows how this algorithm performs on well-scaled and badly scaled quadratics. In the final section, we present the Nelder-Mead variable shape unconstrained optimization algorithm. Several numerical experiments are presented, where some of these are counter examples, that is cases where the algorithms fails to converge on a stationnary point. In the appendix of this document, the interested reader will find a bibliography of simplex-based algorithms, along with an analysis of the various implementations which are available in several programming languages.

Version 0.3
September 2009

Contents

Chapter 1

Introduction

The Nelder-Mead simplex algorithm, published in 1965, is an enormously popular search method for multidimensional unconstrained optimization. The Nelder-Mead algorithm should not be confused with the (probably) more famous simplex algorithm of Dantzig for linear programming. The Nelder-Mead algorithm is especially popular in the fields of chemistry, chemical engineering, and medicine. Two measures of the ubiquity of the Nelder-Mead algorithm are that it appears in the best-selling handbook Numerical Recipes and in Matlab. In [39], Virginia Torczon writes: "Margaret Wright has stated that over fifty percent of the calls received by the support group for the NAG software library concerned the version of the Nelder-Mead simplex algorithm to be found in that library". No derivative of the cost function is required, which makes the algorithm interesting for noisy problems.

The Nelder-Mead algorithm falls in the more general class of direct search algorithms. These methods use values of f taken from a set of sample points and use that information to continue the sampling. The Nelder-Mead algorithm maintains a simplex which are approximations of an optimal point. The vertices are sorted according to the objective function values. The algorithm attemps to replace the worst vertex with a new point, which depends on the worst point and the centre of the best vertices.

The goal of this toolbox is to provide a Nelder-Mead (1965) direct search optimization method to solve the following unconstrained optimization problem

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

where $x \in \mathbb{R}^n$, n is the number of optimization parameters and f is the objective function $f : \mathbb{R}^n \to \mathbb{R}$. In order to solve the unconstrained optimization problem, the Nelder-Mead algorithm uses a variable shape simplex. The toolbox also provide Spendley's et al. algorithm (1962), which uses a fixed shape simplex. Historically, the algorithm designed by Nelder and Mead was designed as an improvement on Spendley's et al. algorithm. The Box complex algorithm (1965), which is an extension of Spendley's et al. algorithm, solves the following constrained problem

$$\min f(x) \tag{1.2}$$

$$\ell_i \leq x_i \leq u_i, \qquad i = 1, n \tag{1.3}$$
$$g_j(x) \geq 0, \qquad j = 1, m \tag{1.4}$$
$$\tag{1.5}$$

where m is the number of nonlinear, positive constraints and $\ell_i, u_i \in \mathbb{R}^n$ are the lower and upper bounds of the variables.

The Nelder-Mead algorithm may be used in the following optimization context :

- there is no need to provide the derivatives of the objective function,
- the number of parameters is small (up to 10-20),
- there are bounds and/or non linear constraints.

The internal design of the system is based on the following components :

- ”neldermead” provides various Nelder-Mead variants and manages for Nelder-Mead specific settings, such as the method to compute the initial simplex, the specific termination criteria,
- ”fminsearch” provides a Scilab commands which aims at behaving as Matlab’s fminsearch. Specific terminations criteria, initial simplex and auxiliary settings are automatically configured so that the behaviour of Matlab’s fminsearch is exactly reproduced.
- ”optimset”, ”optimget” provides Scilab commands to emulate their Matlab counterparts.
- ”nmplot” provides a high-level component which provides directly output pictures for Nelder-Mead algorithm.

The current toolbox is based on (and therefore requires) the following toolboxes

- ”optimbase” provides an abstract class for a general optimization component, including the number of variables, the minimum and maximum bounds, the number of non linear inequality constraints, the loggin system, various termination criteria, the cost function, etc...
- ”optimsimplex” provides a class to manage a simplex made of an arbitrary number of vertices, including the computation of a simplex by various methods (axes, regular, Pfeffer’s, randomized bounds), the computation of the size by various methods (diameter, sigma +, sigma-, etc...),

The following is a list of features the Nelder-Mead prototype algorithm currently provides :

- Manage various simplex initializations
  - initial simplex given by user,

  - initial simplex computed with a length and along the coordinate axes,
  - initial regular simplex computed with Spendley et al. formula
  - initial simplex computed by a small perturbation around the initial guess point

- Manage cost function
  - optionnal additionnal argument
  - direct communication of the task to perform : cost function or inequality constraints

- Manage various termination criteria, including maximum number of iterations, tolerance on function value (relative or absolute),
  - tolerance on x (relative or absolute),
  - tolerance on standard deviation of function value (original termination criteria in [3]),
  - maximum number of evaluations of cost function,
  - absolute or relative simplex size,

- Manage the history of the convergence, including
  - history of function values,
  - history of optimum point,
  - history of simplices,
  - history of termination criterias,

- Provide a plot command which allows to graphically see the history of the simplices toward the optimum,
- Provide query features for the status of the optimization process number of iterations, number of function evaluations, status of execution, function value at initial point, function value at optimal point, etc...
- Spendley et al. fixed shaped algorithm,
- Kelley restart based on simplex gradient,
- O'Neill restart based on factorial search around optimum,
- Box-like method managing bounds and nonlinear inequality constraints based on arbitrary number of vertices in the simplex.

Chapter 2

Overview

In this section, we present the main commands of the Nelder-Mead toolbox as well as an example of use.

2.1 How to use the Toolbox

The design of the toolbox is based on the creation of a new token by the *neldermead_new* command. The Nelder-Mead object associated with this token can then be configured with *neldermead_configure* and queried with *neldermead_cget*. To be more specific, the *neldermead_configure* command allows to configure the number of variables, the objective function and the initial guess.

The main command of the toolbox is the *neldermead_search* command, which solves the optimization problem. After an optimization has been performed, the *neldermead_get* command allows to retrieve the optimum $x^\star$, as well as other parameters, such as the number of iterations performed, the number of evaluations of the function, etc...

2.2 An example

In the following example, one searches the minimum of the 2D Rosenbrock function [35], defined by

$$f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2 \tag{2.1}$$

One begins by defining the function "rosenbrock" which computes the Rosenbrock function. The traditionnal initial guess $(-1.2, 1.0)$ is used. The initial simplex is computed along the axes with a length equal to 0.1. The Nelder-Mead algorithm with variable simplex size is used. The verbose mode is enabled so that messages are generated during the algorithm. After the optimization is performed, the optimum is retrieved with quiery features.

```

function y = rosenbrock (x)
```

```
  y = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
endfunction

nm = neldermead_new ();
nm = neldermead_configure(nm,"-x0",[-1.2 1.0]');
nm = neldermead_configure(nm,"-simplex0method","axes");
nm = neldermead_configure(nm,"-simplex0length",0.1);
nm = neldermead_configure(nm,"-method","variable");
nm = neldermead_configure(nm,"-verbose",1);
nm = neldermead_configure(nm,"-function",rosenbrock);
nm = neldermead_search(nm);
xopt = neldermead_get(nm,"-xopt");
fopt = neldermead_get(nm,"-fopt");
historyfopt = neldermead_get(nm,"-historyfopt");
iterations = neldermead_get(nm,"-iterations");
historyxopt = neldermead_get(nm,"-historyxopt");
historysimplex = neldermead_get(nm,"-historysimplex");
fx0 = neldermead_get(nm,"-fx0");
status = neldermead_get(nm,"-status");
nm = neldermead_destroy(nm);
```

The script makes the hypothesis that an environment variable named TOOLBOX_HOME contains the path to directory which contains the toolbox, which is stored in the "neldermead" directory.

For a deeper presentation of the commands and options, the reader should consult the help which is provided with the package.

Chapter 3

Simplex theory

In this section, we present the various definitions connected to simplex algorithms. We introduce several methods to measure the size of a simplex, including the oriented length. We present several methods to compute an initial simplex, for example the regular simplex used by Spendley et al..

3.1 The simplex

A *simplex* S in $\mathbb{R}^n$ is the convex hull of $n+1$ points $S = \{\mathbf{x}_i\}_{i=1,n+1}$.

Box extended the Nelder-Mead algorithm to handle bound and non linear constraints [4]. To be able to manage difficult cases, he uses a *complex* made of $k \geq n+1$ vertices. In this section, we will state clearly when the definition and results can be applied to a complex. Indeed, some definitions such as the simplex gradient cannot be extended to a *complex* and are only applicable to a *simplex*.

The point $\mathbf{x}_i \in \mathbb{R}^n$ is the i-th vertex of S. Given a function $f(\mathbf{x}) \in \mathbb{R}$, each vertex is associated with a function value $f_i = f(\mathbf{x}_i)$ for $i = 1, n+1$. In simplex algorithms, the vertex are sorted by increasing function values

$$f_1 \leq f_2 \leq \ldots \leq f_n \leq f_{n+1} \tag{3.1}$$

The sorting order is not precisely defined neither in Spendley's et al paper [37] nor in Nelder and Mead's [25]. In [16], the sorting rules are defined precisely to be able to state a theoretical convergence result. In practical implementations, though, the ordering rules have no measurable influence.

Let V denote the $n \times n$ matrix of simplex directions

$$V(S) = (\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1, \ldots, \mathbf{x}_{n+1} - \mathbf{x}_1) = (\mathbf{v}_1, \ldots, \mathbf{v}_n) \tag{3.2}$$

We say that S is nonsingular if the matrix of simplex directions $V(S)$ is nonsingular.

3.2 The size of the simplex

Several methods are available to compute the size of a simplex. In Kelley's book [15], the author presents the diameter and the two oriented lengths.

The simplex diameter $diam(S)$ is defined by

$$diam(S) = \max_{i,j=1,n+1} \|\mathbf{x}_i - \mathbf{x}_j\|_2, \tag{3.3}$$

where $\|.\|_2$ is the euclidian norm $\|x\|_2 = \sum_{i=1,n} \mathbf{x}_i^2$. In practical implementations, computing the diameter requires two nested loops over the vertices of the simplex, i.e. $(n+1)^2$ operations. This is why authors generally prefer to use lengths which are less expensive to compute.

The two oriented lengths $\sigma_-(S)$ and $\sigma_+(S)$ are using the first vertex as the reference point and are defined by

$$\sigma_+(S) = \max_{i=2,n+1} \|\mathbf{x}_i - \mathbf{x}_1\|_2 \qquad \text{and} \qquad \sigma_-(S) = \min_{i=2,n+1} \|\mathbf{x}_i - \mathbf{x}_1\|_2 \tag{3.4}$$

The following inequalities are satisfied between the diameter and the maximum oriented length

$$\sigma_+(S) \leq diam(S) \leq 2\sigma_+(S) \tag{3.5}$$

In Nash's book [21], the size of the simplex $s_N(S)$ is measured based on the $l1$ norm and is defined by

$$s_N(S) = \sum_{i=2,n+1} \|\mathbf{x}_i - \mathbf{x}_1\|_1 \tag{3.6}$$

where

$$\|\mathbf{x}_i - \mathbf{x}_1\|_1 = \sum_{j=1,n} |x_i^j - x_1^j| \tag{3.7}$$

where $x_i^j \in \mathbb{R}$ is the j-th coordinate of the i-th vertex of the simplex S.

3.3 The initial simplex

While most of the theory can be developed without being very specific about the initial simplex, the initial simplex plays a very important role in practice. All approaches are based on the initial guess $\overline{\mathbf{x}}_0 \in \mathbb{R}^n$ and create a geometric shape based on this point. (We denoted the initial guess by $\overline{\mathbf{x}}_0$ instead of the usual $\mathbf{x}_0$ in order to distinguish the initial guess from the vertices $\{\mathbf{x}_i\}_{i=1,n+1}$.)

In this section, we present the various approach to design the initial simplex. In the first part, we emphasize the importance of the initial simplex in optimization algorithms. Then we present the regular simplex approach by Spendley et al., the randomized bounds approach by Box and Pfeffer's method.

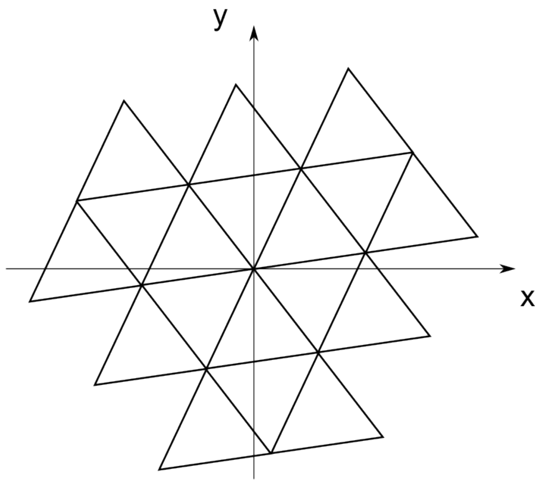

Fig. 3.1 : Typical pattern with fixed-shape Spendley's et al algorithm

3.3.1 Importance of the initial simplex

The initial simplex is particularily important in the case of Spendley's et al method, where the shape of the simplex is fixed during the iterations. Therefore, the algorithm can only go through points which are on the pattern defined by the initial simplex. The pattern presented in figure 3.1 is typical a fixed-shape simplex algorithm (see [39], chapter 3, for other patterns of a direct search method). If, by chance, the pattern is so that the optimum is close to one point defined by the pattern, the number of iteration may be small. On the contrary, the number of iterations may be high if the pattern does not come close to the optimum.

The variable-shape simplex algorithm designed by Nelder and Mead is also very sensitive to the initial simplex. One of the problems is that the initial simplex should be consistently scaled with respect to the unknown x. In [29], "An investigation into the efficiency of variants on the simplex method", Parkinson and Hutchinson explored several ways of improvement. First, they investigate the sensitivity of the algorithm to the initial simplex. Two parameters were investigated, i.e. the initial length and the orientation of the simplex. The conclusion of their study with respect to the initial simplex is the following. "The orientation of the initial simplex has a significant effect on efficiency, but the relationship can be too sensitive for an automatic predictor to provide sufficient accuracy at this time."

Since no initial simplex clearly improves on the others, in practice, it may be convenient to try different approaches.

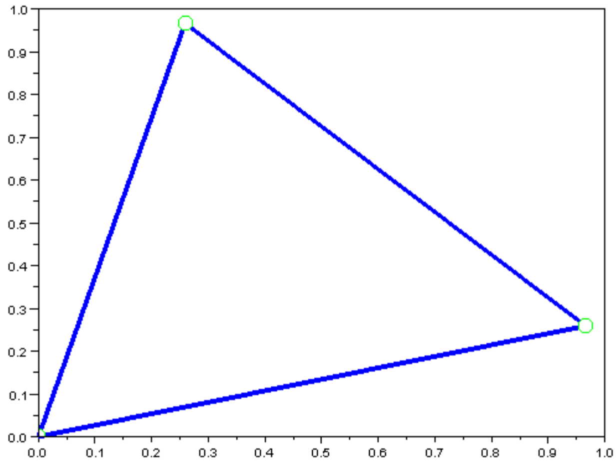

Fig. 3.2 : Regular simplex in 2 dimensions

3.3.2 Spendley's et al simplex

In their paper [37], Spendley et al. use a regular simplex with given size $\ell > 0$. We define the parameters $p, q > 0$ as

$$p = \frac{1}{n\sqrt{2}}\left(n - 1 + \sqrt{n+1}\right), \tag{3.8}$$

$$q = \frac{1}{n\sqrt{2}}\left(\sqrt{n+1} - 1\right). \tag{3.9}$$

We can now define the vertices of the simplex $S = \{\mathbf{x}_i\}_{i=1,n+1}$. The first vertex of the simplex is the initial guess

$$\mathbf{x}_1 = \overline{\mathbf{x}}_0. \tag{3.10}$$

The other vertices are defined by $\mathbf{x}_i = (x_i^1, \ldots x_i^n) \in \mathbb{R}^n$ where the coordinates x_i^j are

$$x_i^j = \begin{cases} \overline{x}_0^j + \ell p, & \text{if } j = i - 1, \\ \overline{x}_0^j + \ell q, & \text{if } j \neq i - 1, \end{cases} \tag{3.11}$$

for vertices $i = 2, n+1$ and components $j = 1, n$, where $\ell \in \mathbb{R}$ is the length of the simplex ($\ell > 0$). Notice that this length is the same for all the vertices which keeps the simplex regular.

The regular initial simplex is presented in figure 3.2.

3.3.3 Simplex along the axes

A very efficient and simple approach leads to an axis-by-axis simplex. This simplex depends on a vector of positive lengths $\mathbf{l} \in \mathbb{R}^n$. The first vertex of the simplex is the initial guess

$$\mathbf{x}_1 = \overline{\mathbf{x}}_0. \tag{3.12}$$

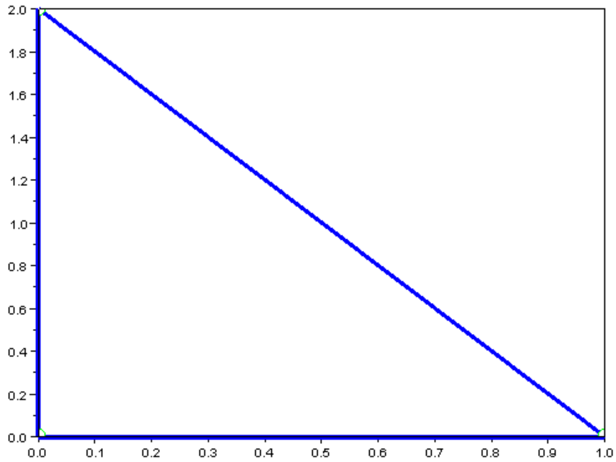

Fig. 3.3 : Axis-based simplex in 2 dimensions

The other vertices are defined by

$$x_i^j = \begin{cases} \overline{x}_0^j + \mathbf{l}_j, \text{ if } j = i-1, \\ \overline{x}_0^j, \text{ if } j \neq i-1, \end{cases} \tag{3.13}$$

for vertices $i = 2, n+1$ and components $j = 1, n$.

This kind of simplex is presented in figure 3.3. The axis-by-axis approach is used in the very popular Nelder-Mead algorithm provided in Numerical Recipes in C [33]. As stated in [33], the length vector $\mathbf{l}$ can be used as a guess for the characteristic length scale of the problem.

3.3.4 Randomized bounds

Assume that the variable $\mathbf{x} \in \mathbb{R}^n$ is bounded so that

$$m^j \leq x^j \leq M^j, \tag{3.14}$$

for $j = 1, n$, where $m_j, M_j \in \mathbb{R}$ are minimum and maximum bounds and $m_j \leq M_j$. A method suggested by Box in [4] is based on the use of pseudo-random numbers. Let $\{\theta_i^j\}_{i=1,n+1,j=1,n} \in [0,1]$ be a sequence of random numbers uniform in the interval $[0,1]$. The first vertex of the simplex is the initial guess

$$\mathbf{x}_1 = \overline{\mathbf{x}}_0. \tag{3.15}$$

The other vertices are defined by

$$x_i^j = m^j + \theta_i^j (M^j - m^j), \tag{3.16}$$

for vertices $i = 2, n+1$ and components $j = 1, n$.

3.3.5 Pfeffer's method

This initial simplex is used in the function *fminsearch* and presented in [7]. It is due to L. Pfeffer at Stanford. The goal of this method is to scale the initial simplex with respect to the characteristic lengths of the problem. This allows, for example, to manage cases where $x_1 \approx 1$ and $x_2 \approx 10^5$. As we are going to see, the scaling is defined with respect to the initial guess $\mathbf{x}_0$. Indeed, the initial simplex is created by small perturbations around the initial guess $\overline{\mathbf{x}}_0$.

The method proceeds by defining $\delta_u, \delta_z > 0$, where δ_u is used for usual components of $\overline{\mathbf{x}}_0$ and δ_z is used for the case where one component of $\overline{\mathbf{x}}_0$ is zero. The default values for δ_u and δ_z are

$$\delta_u = 0.05 \qquad \delta_z = 0.0075. \tag{3.17}$$

The first vertex of the simplex is the initial guess

$$\mathbf{x}_1 \quad = \quad \overline{\mathbf{x}}_0. \tag{3.18}$$

The other vertices are defined by

$$x_i^j \quad = \quad \begin{cases} \overline{x}_0^j + \delta_u \overline{x}_0^j, \text{ if } j = i-1 \text{ and } \overline{x}_0^{j-1} \neq 0, \\ \delta_z, \text{ if } j = i-1 \text{ and } \overline{x}_0^{j-1} = 0, \\ \overline{x}_0^j, \text{ if } j \neq i-1, \end{cases} \tag{3.19}$$

for vertices $i = 2, n+1$ and components $j = 1, n$.

Chapter 4

Spendley's et al. method

In this section, we present Spendley's et al. algorithm [37] for unconstrained optimization.

We begin by presenting a global overview of the algorithm. Then we present various geometric situations which might occur during the algorithm. In the second section, we present several numerical experiments which allow to get some insight of the behaviour of the algorithm on some simple situations. The two first cases are involving only 2 variables and are based on a quadratic function. The last numerical experiment explores the behaviour of the algorith when the number of variables increases.

4.1 Introduction

In this section, we present Spendley's et al algorithm for unconstrained optimization. This algorithm is based on the iterative update of a simplex. At each iteration, either a reflection of a shrink step is performed, so that the shape of the simplex does not change during the iterations. Then we present various geometric situations which might occur during the algorithm. This allows to understand when exactly a reflection or a shrink is performed in practice.

4.1.1 Algorithm

The goal of Spendley's et al. algorithm is to solve the following unconstrained optimization problem

$$\min f(x) \tag{4.1}$$

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

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

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

$$f_1 \leq f_2 \leq \ldots \leq f_n \leq f_{n+1}. \tag{4.2}$$

The *next-to-worst* vertex with index n has a special role in simplex algorithms.

The centroid of the simplex is the center of the vertices where the vertex with index $j = 1, n+1$ has been excluded

$$\overline{x}(j) = \frac{1}{n} \sum_{i=1,n+1,i\neq j} x_i \tag{4.3}$$

The first move of the algorithm is based on the centroid where the worst vertex with index $j = n+1$ has been excluded

$$\overline{x}(n+1) = \frac{1}{n} \sum_{i=1,n} x_i \tag{4.4}$$

The algorithm attemps to replace one vertex x_j by a new point $x(\mu, j)$ between the centroid $\overline{x}$ and the vertex x_j and defined by

$$x(\mu, j) = (1+\mu)\overline{x}(j) - \mu x_j \tag{4.5}$$

The Spendley et al. [37] algorithm makes use of one coefficient, the reflection $\rho > 0$. The standard value of this coefficient is $\rho = 1$.

The first move of the algorithm is based on the reflection with respect to the worst point x_{n+1} so that the reflection point is computed by

$$x(\rho, n+1) = (1+\rho)\overline{x}(n+1) - \rho x_{n+1} \tag{4.6}$$

The algorithm first computes the reflection point with respect to the worst point excluded with $x_r = x(\rho, n+1)$ and evaluates the function value of the reflection point $f_r = f(x_r)$. If that value f_r is better than the worst function value f_{n+1}, the worst point x_{n+1} is rejected from the simplex and the reflection point x_r is accepted. If the reflection point does not improves, the next-to-worst point x_n is reflected and the function is evaluated at the new reflected point. If the function value improves over the worst function value f_{n+1}, the new reflection point is accepted.

At that point of the algorithm, neither the reflection with respect to x_{n+1} nor the reflection with respect to x_n has improved. The algorithm therefore shrinks the simplex toward the best point. That last step uses the shrink coefficient $0 < \sigma < 1$. The standard value for this coefficient is $\sigma = \frac{1}{2}$.

Spendley's et al. algorithm is presented in figure 4.1. The figure 4.2 presents the various moves of the Spendley et al. algorithm. It is obvious from the picture that the algorithm explores a pattern which is entirely determined from the initial simplex.

Compute an initial simplex S_0
Sorts the vertices S_0 with increasing function values
$S \leftarrow S_0$
while $\sigma(S) > tol$ **do**
  $\overline{x} \leftarrow \overline{x}(n+1)$
  $x_r \leftarrow x(\rho, n+1)$, $f_r \leftarrow f(x_r)$ {Reflect with respect to worst}
  if $f_r < f_{n+1}$ **then**
    Accept x_r
  else
    $\overline{x} \leftarrow \overline{x}(n)$
    $x_r \leftarrow x(\rho, n)$, $f_r \leftarrow f(x_r)$ {Reflect with respect to next-to-worst}
    if $f_r < f_{n+1}$ **then**
      Accept x_r
    else
      Compute the points $x_i = x_1 + \sigma(x_i - x_1)$, $i = 2, n+1$ {Shrink}
      Compute the function values at the points $x_i, i = 2, n+1$
    end if
  end if
  Sort the vertices of S with increasing function values
end while

Fig. 4.1 : Spendley et al. algorithm

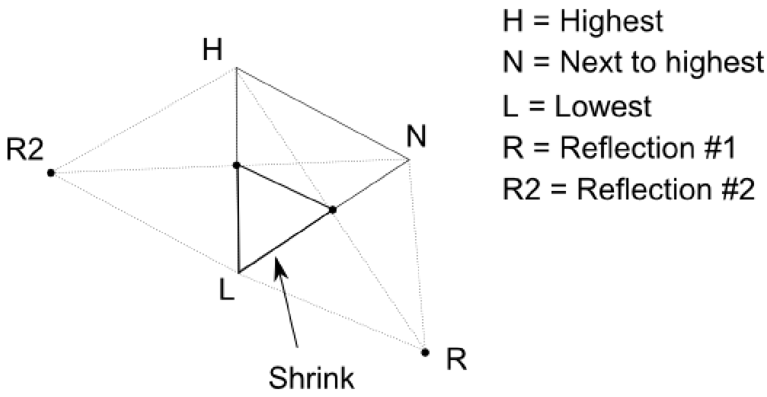

Fig. 4.2 : Spendley et al. simplex moves

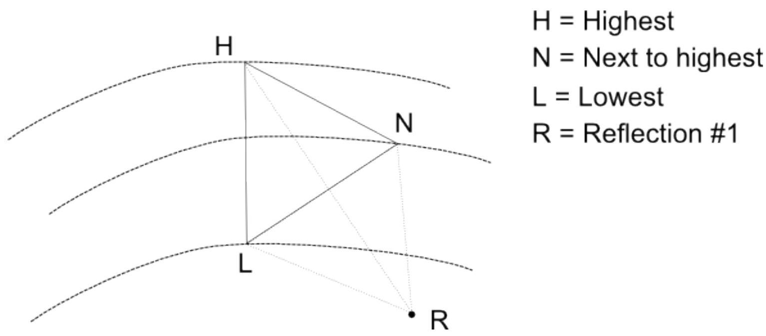

Fig. 4.3 : Spendley et al. simplex moves – Reflection with respect to highest point

4.1.2 Geometric analysis

The figure 4.2 presents the various moves of the simplex in the Spendley et al. algorithm.

The various situations in which these moves are performed are presented in figures 4.3, 4.4 and 4.5.

The basic move is the reflection step, presented in figure 4.3 and 4.4. These two figures shows that the Spendley et al. algorithm is based on a discretization of the parameter space. The optimum is searched on that grid, which is based on regular simplices. When no move is possible to improve the situation on that grid, a shrink step is necessary, as presented in figure 4.5.

In the situation of figure 4.5, neither the reflection #1 or reflection #2 have improved the simplex. Diminishing the size of the simplex by performing a shrink step is the only possible move because the simplex has vertices which are located across the valley. This allows to refine the discretization grid on which the optimum is searched.

4.2 Numerical experiments

In this section, we present some numerical experiments with the Spendley et al. algorithm.

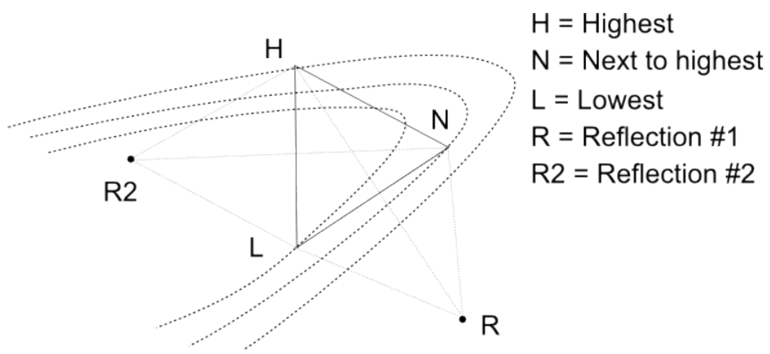

Fig. 4.4 : Spendley et al. simplex moves – Reflection with respect to next-to-highest point. It may happen that the next iteration is a shrink step.

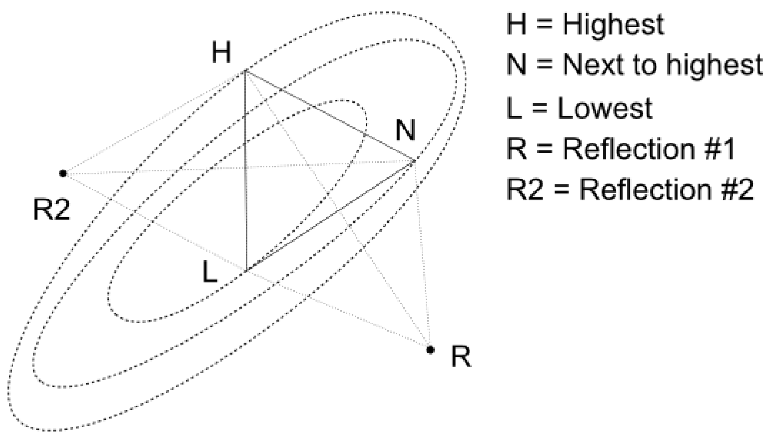

Fig. 4.5 : Spendley et al. simplex moves – Shrink.

4.2.1 Quadratic function

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

$$f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2 \tag{4.7}$$

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

$$\sigma(S) < tol \times \sigma(S_0) \tag{4.8}$$

The initial simplex is a regular simplex with length unity.

The following Scilab script performs the optimization.

```
function y = quadratic (x)
  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","disabled");
nm = neldermead_configure(nm,"-tolsimplexizerelative",1.e-8);
nm = neldermead_configure(nm,"-simplex0method","spendley");
nm = neldermead_configure(nm,"-method","fixed");
nm = neldermead_configure(nm,"-verbose",1);
nm = neldermead_configure(nm,"-verbosetermination",0);
nm = neldermead_search(nm);
neldermead_display(nm);
nm = neldermead_destroy(nm);
```

The numerical results are presented in table 4.6.

Iterations	49
Function Evaluations	132
x_0	$(2.0, 2.0)$
Relative tolerance on simplex size	10^{-8}
Exact $x^\star$	$(0., 0.)$
Computed $x^\star$	$(2.169e-10, 2.169e-10)$
Computed $f(x^\star)$	$4.706e-20$

Fig. 4.6 : Numerical experiment with Spendley's et al. 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 4.7. The method use reflections in the early iterations. Then there is no possible improvement using reflections and shrinking is necessary. That behaviour is an illustration of the discretization which has already been discussed.

The figure 4.8 presents the history of the oriented length of the simplex. The length is updated step by step, where each step corresponds to a shrink in the algorithm.

The convergence is quite fast in this case, since less than 60 iterations allow to get a function value lower than 10^{-15}, as shown in figure 4.9.

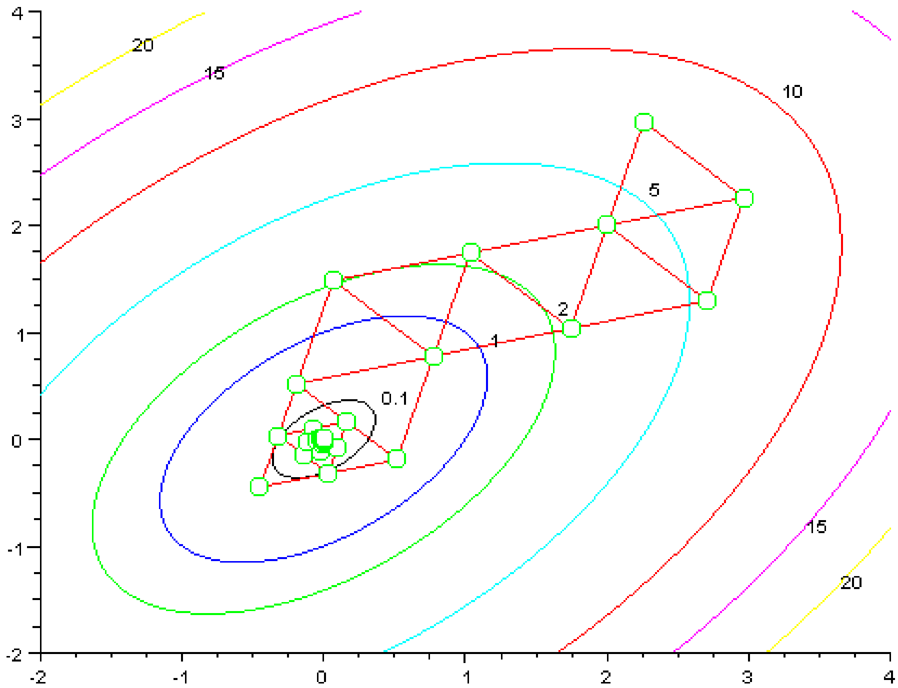

Fig. 4.7 : Spendley et al. numerical experiment – History of simplex

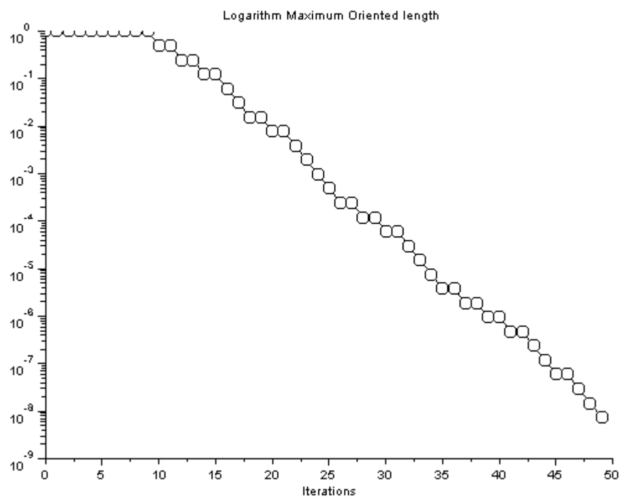

Fig. 4.8 : Spendley et al. numerical experiment – History of logarithm of the size of the simplex

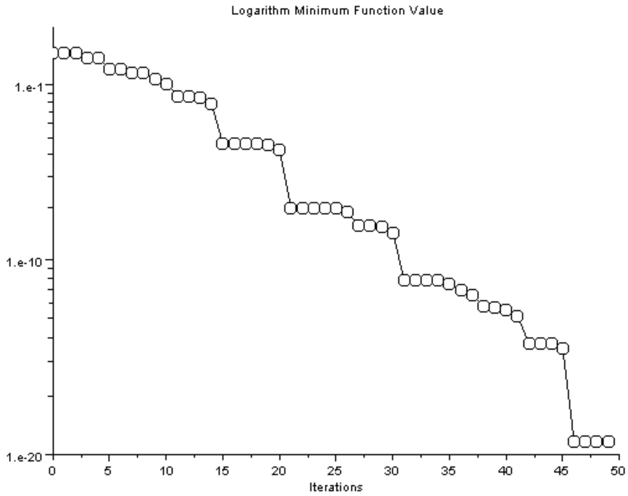

Fig. 4.9 : Spendley et al. numerical experiment – History of logarithm of function

4.2.2 Badly scaled quadratic function

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

$$f(x_1, x_2) = ax_1^2 + x_2^2, \tag{4.9}$$

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 length unity.

The following Scilab script uses the neldermead algorithm to perform the optimization.

```
function y = quadratic (x)
  y = 100 * x(1)^2 + x(2)^2;
endfunction
nm = nmplot_new ();
nm = nmplot_configure(nm,"-numberofvariables",2);
nm = nmplot_configure(nm,"-function",quadratic);
nm = nmplot_configure(nm,"-x0",[10.0 10.0]');
nm = nmplot_configure(nm,"-maxiter",400);
nm = nmplot_configure(nm,"-maxfunevals",400);
nm = nmplot_configure(nm,"-tolxmethod","disabled");
nm = nmplot_configure(nm,"-tolsimplexizerelative",1.e-8);
nm = nmplot_configure(nm,"-simplex0method","spendley");
nm = nmplot_configure(nm,"-method","fixed");
nm = nmplot_configure(nm,"-verbose",1);
nm = nmplot_configure(nm,"-verbosetermination",0);
nm = nmplot_configure(nm,"-simplexfn","rosenbrock.fixed.history.simplex.txt");
nm = nmplot_configure(nm,"-fbarfn","rosenbrock.fixed.history.fbar.txt");
nm = nmplot_configure(nm,"-foptfn","rosenbrock.fixed.history.fopt.txt");
nm = nmplot_configure(nm,"-sigmafn","rosenbrock.fixed.history.sigma.txt");
nm = nmplot_search(nm);
nmplot_display(nm);
nm = nmplot_destroy(nm);
```

The numerical results are presented in table 4.6, where the experiment is presented for $a = 100$. One can check that the number of function evaluation is equal to its maximum limit, even if the value of the function at optimum is very inacurate ($f(x^\star) \approx 0.08$).

Iterations	340
Function Evaluations	400
a	100.0
x_0	$(10.0, 10.0)$
Relative tolerance on simplex size	10^{-8}
Exact $x^\star$	$(0., 0.)$
Computed $x^\star$	$(0.001, 0.2)$
Computed $f(x^\star)$	0.08

Fig. 4.10 : Numerical experiment with Spendley's et al. method on a badly scaled quadratic function

The various simplices generated during the iterations are presented in figure 4.11. The method use reflections in the early iterations. Then there is no possible improvment using reflections and shrinking is necessary. But the shrinking makes the simplex very small so that a large number of iterations are necessary to improve the function value. This is a limitation of the method, which is based on a simplex which can vary its size, but not its shape.

In figure 4.12, we analyse the behaviour of the method with respect to scaling. We check that the method behave poorly when the scaling is bad. The convergence speed is slower and slower and impractical when $a > 10$

4.2.3 Sensitivity to dimension

In this section, we try to study the convergence of the Spendley et al. algorithm with respect to the number of variables. The function we try to minimize is the following quadratic function in n-dimensions

$$f(\mathbf{x}) = \sum_{i=1,n} x_i^2. \tag{4.10}$$

The initial guess is at 0 so that this vertex is never updated during the iterations. The initial simplex is computed with a random number generator. The first vertex of the initial simplex is the origin. The other vertices are uniform in the $[-1, 1]$ interval. for $i = 2, 2, \ldots, n+1$. An absolute termination criteria on the size of the simplex is used. More precisely, the method was stopped when $\sigma(Sk) \leq 10^{-8}$ is satisfied.

For this test, we compute the rate of convergence as presented in Han & Neuman [11]. This rate is defined as

$$\rho(S_0, n) = \lim\ \sup_{k\to\infty} \left(\prod_{i=0,k-1} \frac{\sigma(S_{i+1})}{\sigma(S_i)} \right)^{1/k}, \tag{4.11}$$

where k is the number of iterations. 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

$$\rho(S_0, n) = \lim\ \sup_{k\to\infty} \left(\frac{\sigma(S_k)}{\sigma(S_0)} \right)^{1/k}, \tag{4.12}$$

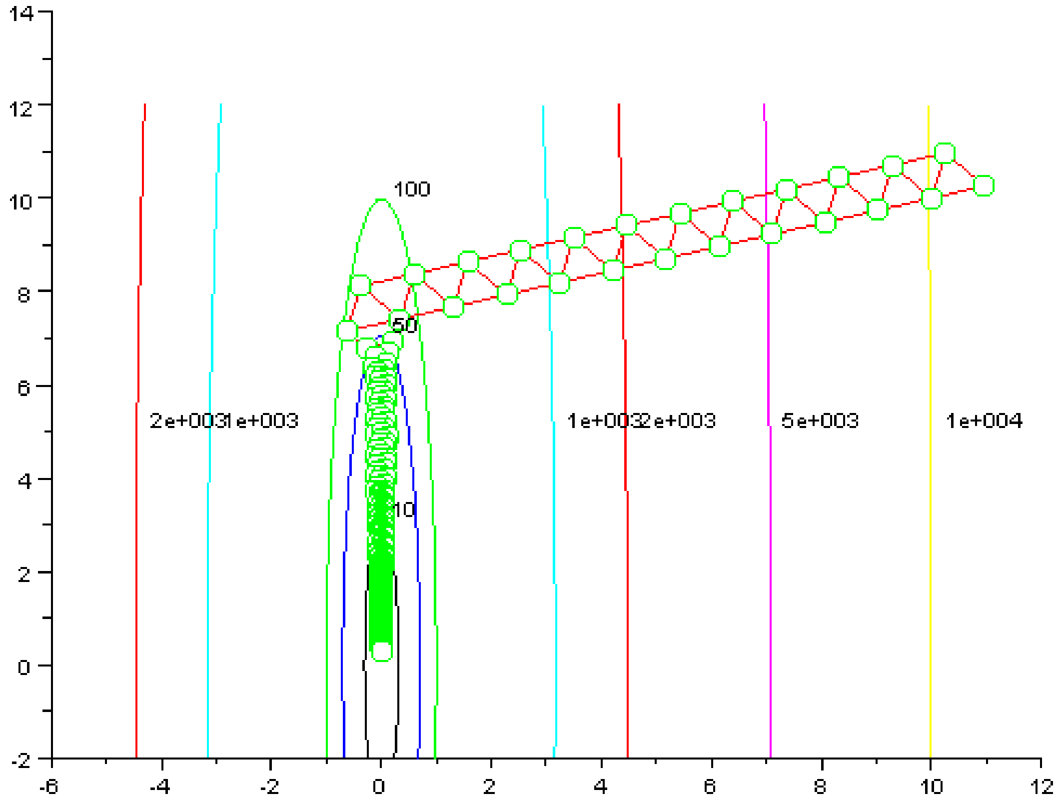

Fig. 4.11 : Spendley et al. numerical experiment with $f(x_1, x_2) = ax_1^2 + x_2^2$ and $a = 100$ – History of simplex

a	Function evaluations	Computed $f(x^\star)$
1.0	160	$2.35e-18$
10.0	222	$1.2e-17$
100.0	400	0.083
1000.0	400	30.3
10000.0	400	56.08

Fig. 4.12 : Numerical experiment with Spendley's et al. method on a badly scaled quadratic function

If k is the number of iterations required to obtain convergence, as indicated by the termination criteria, the rate of convergence is practically computed as

$$\rho(S_0, n, k) = \left(\frac{\sigma(S_k)}{\sigma(S_0)}\right)^{1/k} \tag{4.13}$$

```
function y = quadratic (x)
  y = x(:).' * x(:);
endfunction
//
// myoutputcmd --
//  This command is called back by the Nelder-Mead
//  algorithm.
// Arguments
//  state : the current state of the algorithm
//    "init", "iter", "done"
//  data : the data at the current state
//    This is a tlist with the following entries:
//    * x : the optimal vector of parameters
//    * fval : the minimum function value
//    * simplex : the simplex, as a simplex object
//    * iteration : the number of iterations performed
//    * funccount : the number of function evaluations
//    * step : the type of step in the previous iteration
//
function myoutputcmd ( state , data , step )
  global STEP_COUNTER
  STEP_COUNTER(step) = STEP_COUNTER(step) + 1
endfunction

// OptimizeHanNeumann --
//   Perform the optimization and returns the object
// Arguments
//   N : the dimension
function nm = OptimizeHanNeumann ( N )
  global STEP_COUNTER
  STEP_COUNTER("init") = 0;
  STEP_COUNTER("done") = 0;
  STEP_COUNTER("reflection") = 0;
  STEP_COUNTER("expansion") = 0;
  STEP_COUNTER("insidecontraction") = 0;
  STEP_COUNTER("outsidecontraction") = 0;
  STEP_COUNTER("expansion") = 0;
  STEP_COUNTER("shrink") = 0;
  STEP_COUNTER("reflectionnext") = 0;

  x0 = zeros(N,1);
  nm = neldermead_new ();
  nm = neldermead_configure(nm,"-numberofvariables",N);
  nm = neldermead_configure(nm,"-function",quadratic);
  nm = neldermead_configure(nm,"-x0",x0);
  nm = neldermead_configure(nm,"-maxiter",10000);
  nm = neldermead_configure(nm,"-maxfunevals",10000);
  nm = neldermead_configure(nm,"-tolxmethod","disabled");
  nm = neldermead_configure(nm,"-tolsimplexizeabsolute",1.e-8);
  nm = neldermead_configure(nm,"-tolsimplexizerelative",0);
  nm = neldermead_configure(nm,"-simplex0method","given");
  coords0(1,1:N) = zeros(1,N);
  coords0(2:N+1,1:N) = 2 * rand(N,N) - 1;
  nm = neldermead_configure(nm,"-coords0",coords0);
  nm = neldermead_configure(nm,"-method","fixed");
  nm = neldermead_configure(nm,"-verbose",0);
  nm = neldermead_configure(nm,"-verbosetermination",0);
  nm = neldermead_configure(nm,"-outputcommand",myoutputcmd);
  //
  // Perform optimization
  //
  nm = neldermead_search(nm);
endfunction

for N = 1:10
  nm = OptimizeHanNeumann ( N );
  niter = neldermead_get ( nm , "-iterations" );
  funevals = neldermead_get ( nm , "-funevals" );
  simplex0 = neldermead_get ( nm , "-simplex0" );
  sigma0 = optimsimplex_size ( simplex0 , "sigmaplus" );
```

```
    simplexopt = neldermead_get ( nm , "-simplexopt" );
    sigmaopt = optimsimplex_size ( simplexopt , "sigmaplus" );
    rho = ( sigmaopt / sigma0 ) ^ ( 1 / niter );
    //mprintf ( "%d %d %d %e\n" , N , funevals , niter , rho );
    mprintf("%d %s\n",N, strcat(string(STEP_COUNTER)," "))
   nm = neldermead_destroy(nm);
end
```

The figure 4.13 presents the type of steps which are performed for each number of variables. We see that the algorithm mostly performs shrink steps.

n	#Iterations	# Reflections / High	# Reflection / Next to High	#Shrink
1	27	0	0	26
2	28	0	0	27
3	30	2	0	27
4	31	1	1	28
5	29	0	0	28
6	31	2	0	28
7	29	0	0	28
8	29	0	0	28
9	29	0	0	28
10	29	0	0	28
11	29	0	0	28
12	29	0	0	28
13	31	0	2	28
14	29	0	0	28
15	29	0	0	28
16	31	0	1	29
17	30	0	0	29
18	30	0	0	29
19	31	0	1	29
20	32	2	0	29

Fig. 4.13 : Numerical experiment with Spendley et al method on a generalized quadratic function – number and kinds of steps performed

The figure 4.14 presents the number of function evaluations depending on the number of variables.

One can see that the number of function evaluations increases approximately linearily with the dimension of the problem in figure 4.15. A rough rule of thumb is that, for $n = 1, 20$, the number of function evaluations is equal to $30n$. This test is in fact the best that we can expect from this algorithm : since most iterations are shrink steps, most iterations improves the function value.

The table 4.14 also shows the interesting fact that the convergence rate is almost constant and very close to 1. This is a consequence of the shrink steps, which are dividing the size of the simplex at each iterations by 2. Therefore, the convergence rate is close to 1/2.

4.3 Conclusion

We saw in the first numerical experiment that the method behave reasonably when the function is correctly scaled. When the function is badly scaled, as in the second numerical experiment, the Spendley et al. algorithm produces a large number of function evaluations and converges very slowly. This limitation occurs with even moderate badly scaled functions and generates a very slow method in these cases.

n	Function Evaluations	Iterations	$\rho(S_0, n)$
1	81	27	0.513002
2	112	28	0.512532
3	142	29	0.524482
4	168	28	0.512532
5	206	31	0.534545
6	232	29	0.512095
7	262	30	0.523127
8	292	30	0.523647
9	321	30	0.523647
10	348	29	0.512095
11	377	29	0.512095
12	406	29	0.512095
13	435	29	0.512095
14	464	29	0.512095
15	493	29	0.512095
16	540	30	0.511687
17	570	30	0.511687
18	600	30	0.511687
19	630	30	0.511687
20	660	30	0.511687

Fig. 4.14 : Numerical experiment with Spendley et al. method on a generalized quadratic function

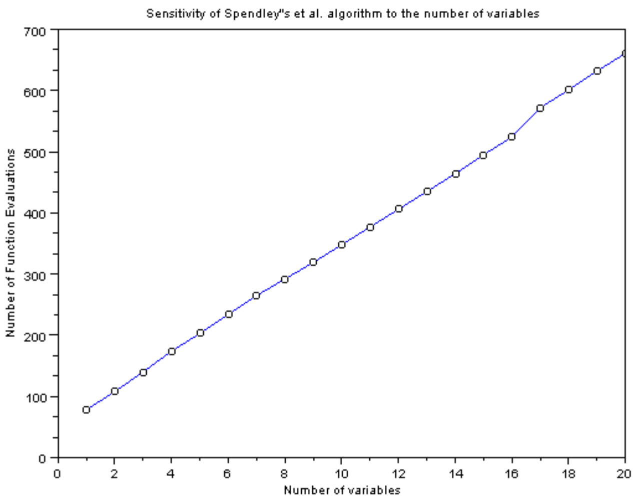

Fig. 4.15 : Spendley et al. numerical experiment – Number of function evaluations depending on the number of variables

In the last experiment, we have explored what happens when the number of iterations is increasing. The rate of convergence in this case is close to $1/2$ and the number of function evaluations is a linear function of the number of variables (approximately $30n$ in this case).

Chapter 5

Nelder-Mead method

In this chapter, we present Nelder and Mead's [25] 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 might be 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 [11], which is based on 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 analyse the behaviour of the algorithm on quadratic functions and present several counter examples where the Nelder-Mead algorithm is known to fail.

5.1 Introduction

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

$$\min f(x) \tag{5.1}$$

where $x \in \mathbb{R}^n$, 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 improvment over the Spendley's et al. method with the goal of allowing the simplex to vary in shape. 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" [25].

These parameters should satisfy the following inequalities [25, 16]

$$\rho > 0, \qquad \chi > 1, \qquad \chi > \rho, \qquad 0 < \gamma < 1, \qquad \text{and} \qquad 0 < \sigma < 1. \tag{5.2}$$

The standard values for these coefficients are

$$\rho = 1, \qquad \chi = 2, \qquad \gamma = \frac{1}{2}, \qquad \text{and} \qquad \sigma = \frac{1}{2}. \tag{5.3}$$

In [15], 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

$$-1 < \mu_{ic} < 0 < \mu_{oc} < \mu_r < \mu_e \tag{5.4}$$

The Nelder-Mead algorithm is presented in figure 5.1.

The algorithm from figure 5.1 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 [27] and the corresponding algorithm is presented in figure 5.2.

5.2 Geometric analysis

The figure 5.3 presents the various moves of the simplex in the Nelder-Mead algorithm.

The figure 5.4 through 5.9 present the detailed situations when each kind of step occur.

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.

When the simplex is near the optimum, the inside and outside contraction steps may be performed, which allow to decrease the size of the simplex.

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.

5.3 Convergence properties on a quadratic

In this section, we reproduce one result presented by Han and Neumann [11], which states the rate of convergence toward the optimum on a class of quadratic functions with a special initial simplex. Some additionnal results are also presented in the Phd thesis by Lixing Han in 2000 [10]. 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 caracteristic equation. The study of the roots of these caracteristic equations give an insight of the behaviour of the Nelder-Mead algorithm when the dimension n increases.

Let us suppose than one wants to minimize the function

$$f(\mathbf{x}) = x_1^2 + \ldots + x_n^2 \tag{5.5}$$

with the initial simplex

$$S_0 = \left[\mathbf{0}, \mathbf{v}_1^{(0)}, \ldots, \mathbf{v}_n^{(0)}\right] \tag{5.6}$$

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

Fig. 5.1 : Nelder-Mead algorithm - standard version

if $f_e < f_1$ **then**
  Accept x_e
else
  Accept x_r
end if

Fig. 5.2 : Nelder-Mead algorithm - greedy version

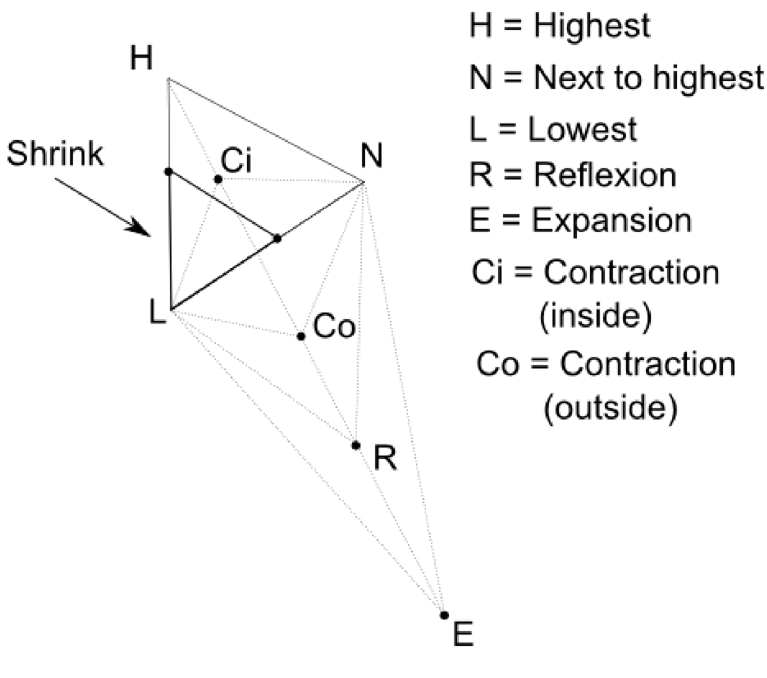

Fig. 5.3 : Nelder-Mead simplex moves

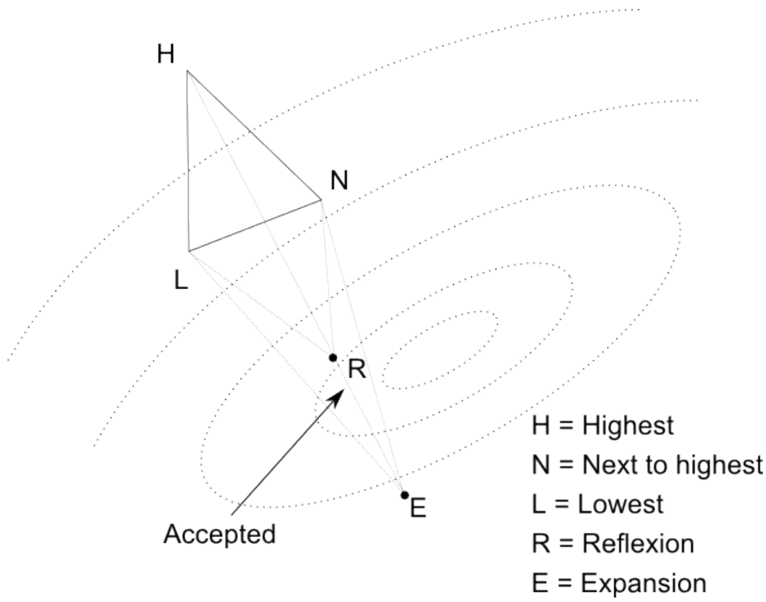

Fig. 5.4 : Nelder-Mead simplex moves - reflection

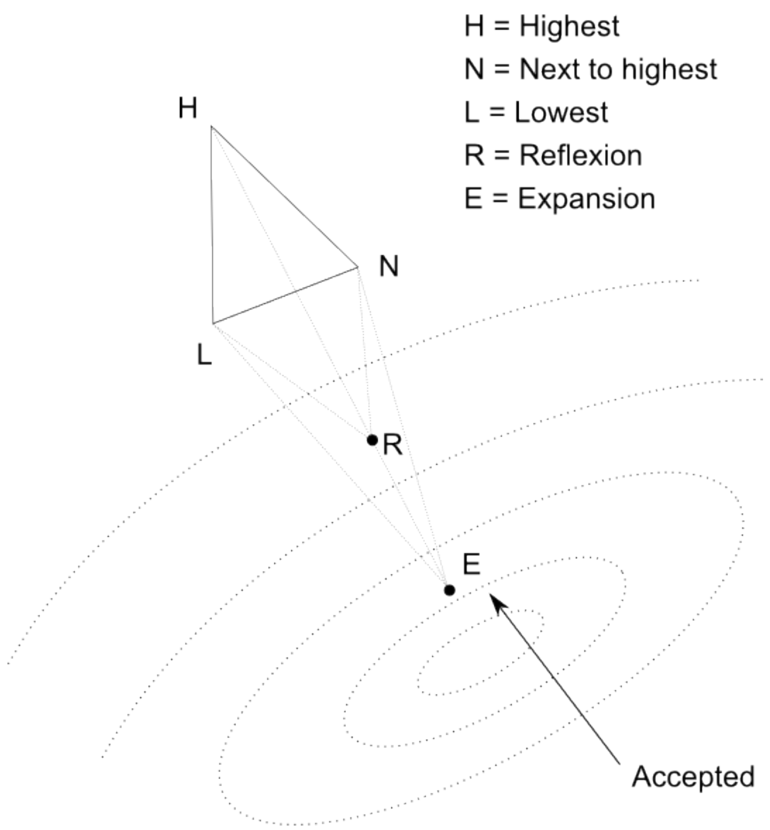

Fig. 5.5 : Nelder-Mead simplex moves - expansion

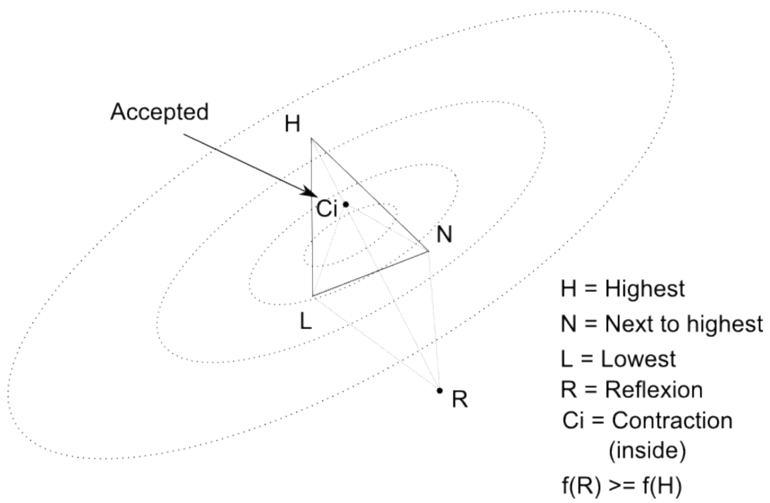

Fig. 5.6 : Nelder-Mead simplex moves - inside contraction

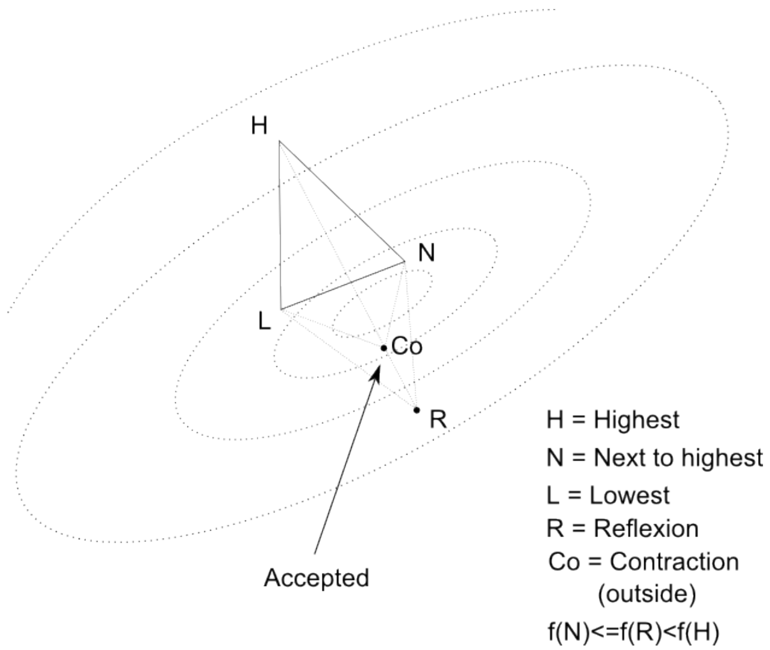

Fig. 5.7 : Nelder-Mead simplex moves - outside contraction

With this choice of the initial simplex, the best vertex remains fixed at $\mathbf{0} \in \mathbb{R}^n$. As the function in equation 5.5 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

$$S_k = \left[\mathbf{0}, \mathbf{v}_1^{(k)}, \ldots, \mathbf{v}_n^{(n)}\right] \tag{5.7}$$

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 of the simplex S_k, $\sigma(S_k)$. We say that a sequence of simplices S_k converges to the minimizer $\mathbf{0} \in \mathbb{R}^n$ of the function in equation 5.5 if $\lim_{k\to\infty} \sigma(S_k) = 0$.

We measure the rate of convergence defined by [11]

$$\rho(S_0, n) = \lim\ \sup_{k\to\infty} \left(\sum_{i=0,k-1} \frac{\sigma(S_{i+1}}{\sigma(S_i} \right)^{1/k} \tag{5.8}$$

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

$$\rho(S_0, n) = \lim\ \sup_{k\to\infty} \left(\frac{\sigma(S_{k+1}}{\sigma(S_0} \right)^{1/k} \tag{5.9}$$

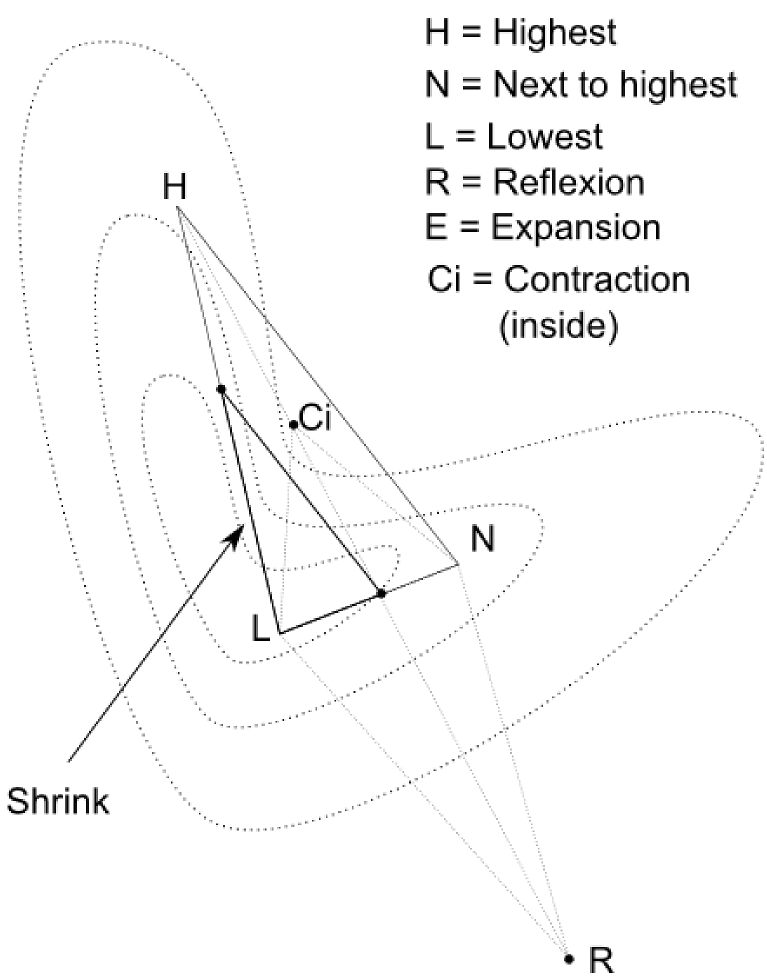

Fig. 5.8 : Nelder-Mead simplex moves - shrink after inside contraction.

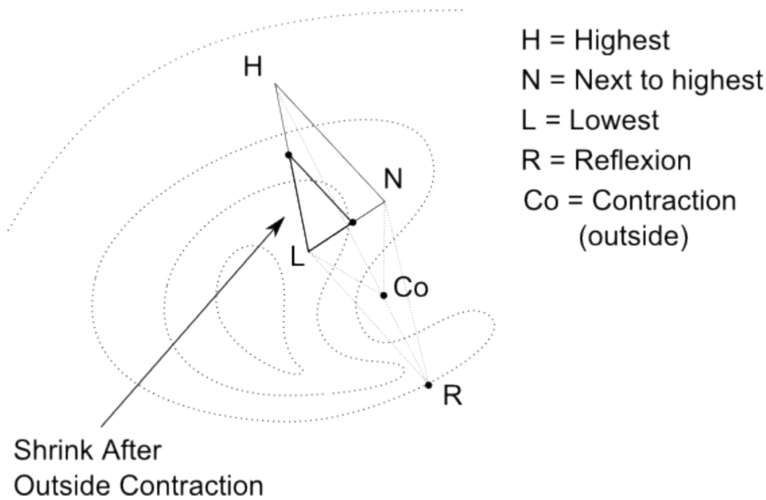

Fig. 5.9 : Nelder-Mead simplex moves - shrink after outside contraction

According to the definition, the algorithm is convergent if $0 \leq \rho(S_0, n) < 1$. 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

$$S_k = \left[\mathbf{0}, \mathbf{v}^{(k+n-1)}, \ldots, \mathbf{v}^{(k+1)}, \mathbf{v}^{(k)}\right] \tag{5.10}$$

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

$$f(\mathbf{0}) < f(\mathbf{v}^{(k+n-1)} < f(\mathbf{v}^{(k+1)}) < f(\mathbf{v}^{(k)}), \tag{5.11}$$

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 establich recurrence equations for the sucessive 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

$$\begin{aligned} \overline{\mathbf{v}}^{(k)} &= \frac{1}{n} \sum_{i=1,n-1} \mathbf{v}^{(k+i)} && (5.12) \\ &= \frac{1}{n} \left(\mathbf{v}^{(k+1)} + \ldots + \mathbf{v}^{(k+n-1)}\right) && (5.13) \end{aligned}$$

5.3.1 With default parameters

In this section, we analyse the roots of the caracteristic 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

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} + \frac{1}{2}\left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right) \tag{5.14}$$

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

$$2n\mathbf{v}^{(k+n)} - 3\mathbf{v}^{(k+1)} - \ldots - 3\mathbf{v}^{(k+n-1)} + n\mathbf{v}^{(k)} = 0 \tag{5.15}$$

The associated caracteristic equation is

$$2n\mu^n - 3\mu^{n-1} - \ldots - 3\mu + n = 0. \tag{5.16}$$

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

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} - \frac{1}{2}\left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right) \tag{5.17}$$

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

$$2n\mathbf{v}^{(k+n)} - \mathbf{v}^{(k+1)} - \ldots - \mathbf{v}^{(k+n-1)} - n\mathbf{v}^{(k)} = 0 \tag{5.18}$$

The associated caracteristic equation is

$$2n\mu^n - \mu^{n-1} - \ldots - \mu - n = 0. \tag{5.19}$$

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

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} + \left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right) \tag{5.20}$$

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

$$n\mathbf{v}^{(k+n)} - 2\mathbf{v}^{(k+1)} - \ldots - 2\mathbf{v}^{(k+n-1)} + n\mathbf{v}^{(k)} = 0 \tag{5.21}$$

The associated caracteristic equation is

$$n\mu^n - 2\mu^{n-1} - \ldots - 2\mu + n = 0. \tag{5.22}$$

The recurrence equations 5.16, 5.19 and 5.22 are linear. Their general solutions are of the form

$$\mathbf{v}^{(k)} = \mu_1^k \mathbf{a}_1 + \ldots + \mu_n^k \mathbf{a}_n \tag{5.23}$$

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 [11] 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 [11] perform a numerical analysis of the roots.

In the following Scilab script, one computes the roots of these 3 caracteristic equations.

```
//
// computeroots1 --
//   Compute the roots of the caracteristic 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)
  r = roots(p)
  for i=1:n
    mprintf("#%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)
  r = roots(p)
  for i=1:n
    mprintf("#%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)
  for i=1:n
    mprintf("#%d/%d |%s|=%f\n", i, length(r),string(r(i)),abs(r(i)))
  end
endfunction
```

If one executes this 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
#1/10 |0.5822700+%i*0.7362568|=0.938676
#2/10 |0.5822700-%i*0.7362568|=0.938676
#3/10 |-0.5439060+%i*0.7651230|=0.938747
#4/10 |-0.5439060-%i*0.7651230|=0.938747
#5/10 |0.9093766+%i*0.0471756|=0.910599
#6/10 |0.9093766-%i*0.0471756|=0.910599
#7/10 |0.0191306+%i*0.9385387|=0.938734
#8/10 |0.0191306-%i*0.9385387|=0.938734
#9/10 |-0.8918713+%i*0.2929516|=0.938752
#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
#1/10 |0.7461586+%i*0.5514088|=0.927795
#2/10 |0.7461586-%i*0.5514088|=0.927795
#3/10 |-0.2879931+%i*0.8802612|=0.926175
#4/10 |-0.2879931-%i*0.8802612|=0.926175
#5/10 |-0.9260704|=0.926070
#6/10 |0.9933286|=0.993329
#7/10 |0.2829249+%i*0.8821821|=0.926440
#8/10 |0.2829249-%i*0.8821821|=0.926440
#9/10 |-0.7497195+%i*0.5436596|=0.926091
#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
#1/10 |0.6172695+%i*0.7867517|=1.000000
#2/10 |0.6172695-%i*0.7867517|=1.000000
#3/10 |-0.5801834+%i*0.8144859|=1.000000
#4/10 |-0.5801834-%i*0.8144859|=1.000000
#5/10 |0.9946011+%i*0.1037722|=1.000000
#6/10 |0.9946011-%i*0.1037722|=1.000000
#7/10 |0.0184670+%i*0.9998295|=1.000000
#8/10 |0.0184670-%i*0.9998295|=1.000000
#9/10 |-0.9501543+%i*0.3117800|=1.000000
#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 computationnal 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 caracteristic equation roots";
  f.children.x_label.text = "Number of variables (n)";
  f.children.y_label.text = "Roots of the caracteristic 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 5.10. These roots are presented graphically in figure 5.11. One can see that the roots converge toward 1 when $n \to \infty$.

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

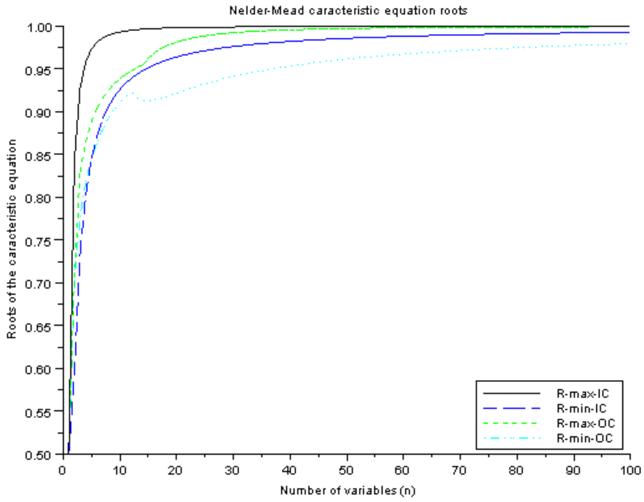

Fig. 5.11 : Modulus of the roots of the caracteristic equations of the Nelder-Mead method with standard coefficients – R-max-IC is the maximum of the modulus of the root of the Inside Contraction steps

5.3.2 With variable parameters

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

Outside contraction If the outside contraction step is repeatedly performed with variable $\mu_{oc} > 0$, then

$$\mathbf{v}^{(k+n)} = \overline{\mathbf{v}}^{(k)} + \mu_{oc}\left(\overline{\mathbf{v}}^{(k)} - \mathbf{v}^{(k)}\right) \quad (5.24)$$

$$= (1+\mu_{oc})\overline{\mathbf{v}}^{(k)} - \mu_{oc}\mathbf{v}^{(k)} \quad (5.25)$$

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

$$n\mathbf{v}^{(k+n)} - (1+\mu_{oc})\mathbf{v}^{(k+1)} - \ldots - (1+\mu_{oc})\mathbf{v}^{(k+n-1)} + n\mu_{oc}\mathbf{v}^{(k)} = 0 \quad (5.26)$$

The associated caracteristic equation is

$$n\mu^n - (1+\mu_{oc})\mu^{n-1} - \ldots - (1+\mu_{oc})\mu + n\mu_{oc} = 0. \quad (5.27)$$

Inside contraction We suppose that the inside contraction step is repeatedly performed with $-1 < \mu_{ic} < 0$. The caracteristic equation is the same as 5.27, but it is here studied in the range $\mu_{ic} \in]-1, 0[$.

To study the convergence of the method, one simply have to study the roots of equation 5.27, 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, one computes the minimum and maximum root of the caracteristic equation, with n fixed.

```
//
// rootsvariable --
//   Compute roots of the caracteristic 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 caracteristic equation roots";
  f.children.x_label.text = "Contraction coefficient";
  f.children.y_label.text = "Roots of the caracteristic 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;
```

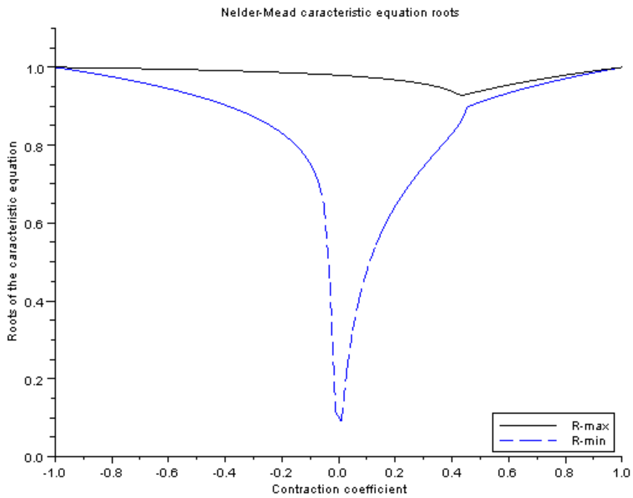

Fig. 5.12 : Modulus of the roots of the caracteristic equations of the Nelder-Mead method with variable contraction coefficient and $n = 10$ – R-max is the maximum of the modulus of the root of the caracteristic equation

```
  mypoly.line_style=i;
  end
  xs2png(0,"neldermead-roots-variable.png");
endfunction
```

The figure 5.12 presents the minimum and maximum modulus of the roots of the caracteristic 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.

5.4 Numerical experiments

In this section, we present some numerical experiments with the Nelder-Mead algorithm.

5.4.1 Quadratic function

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

$$f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2 \tag{5.28}$$

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

$$\sigma(S) < tol \times \sigma(S_0) \tag{5.29}$$

The initial simplex is computed from the coordinate axis and the unit length. The numerical results are presented in table 5.13.

Iterations	65
Function Evaluations	127
x_0	$(2.0, 2.0)$
Relative tolerance on simplex size	10^{-8}
Exact $x^\star$	$(0., 0.)$
Computed $x^\star$	$(7.3e-10, -2.5e-9)$
Computed $f(x^\star)$	$8.7e-18$

Fig. 5.13 : 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 5.14. The method use reflections in the early iterations. Then there is no possible improvment using reflections and shrinking is necessary.

The figure 5.15 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 60 iterations allow to get a function value lower than 10^{-15}, as shown in figure 5.16.

Badly scaled quadratic function

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

$$f(x_1, x_2) = a x_1^2 + x_2^2, \tag{5.30}$$

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 computed from the coordinate axis and the unit length.

The numerical results are presented in table 5.17, where the experiment is presented for $a = 100$. One 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

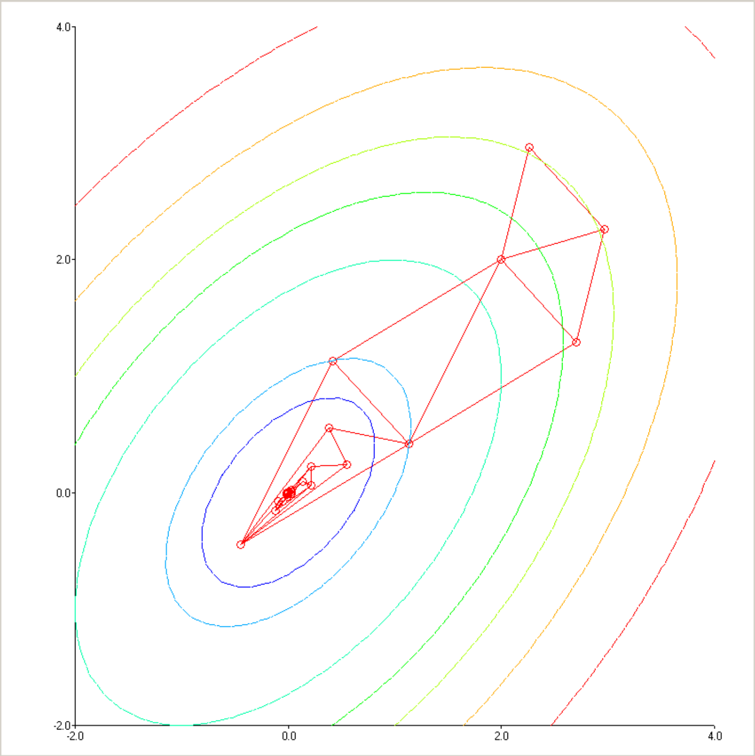

Fig. 5.14 : Nelder-Mead numerical experiment – history of simplex

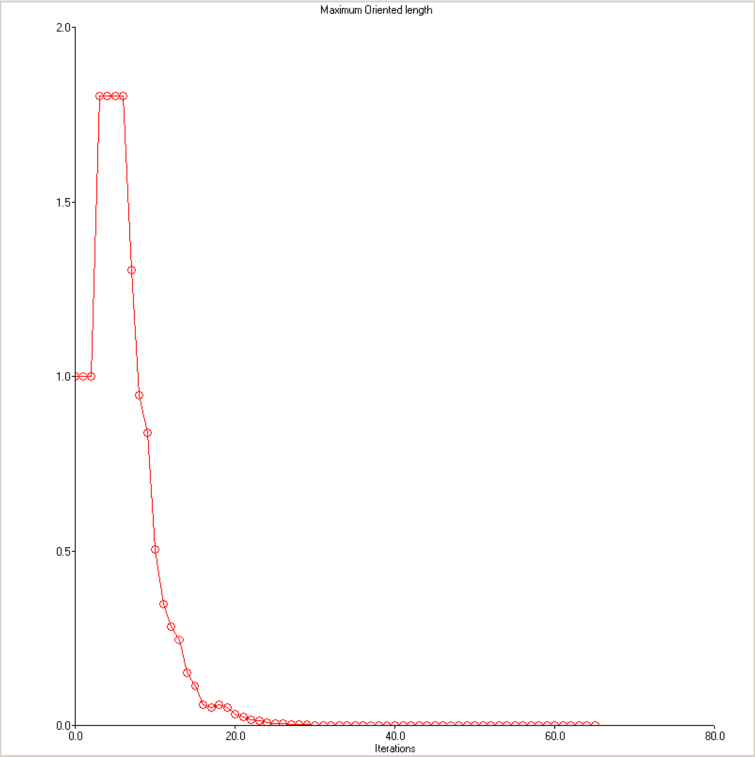

Fig. 5.15 : Nelder-Mead numerical experiment – history of length of simplex

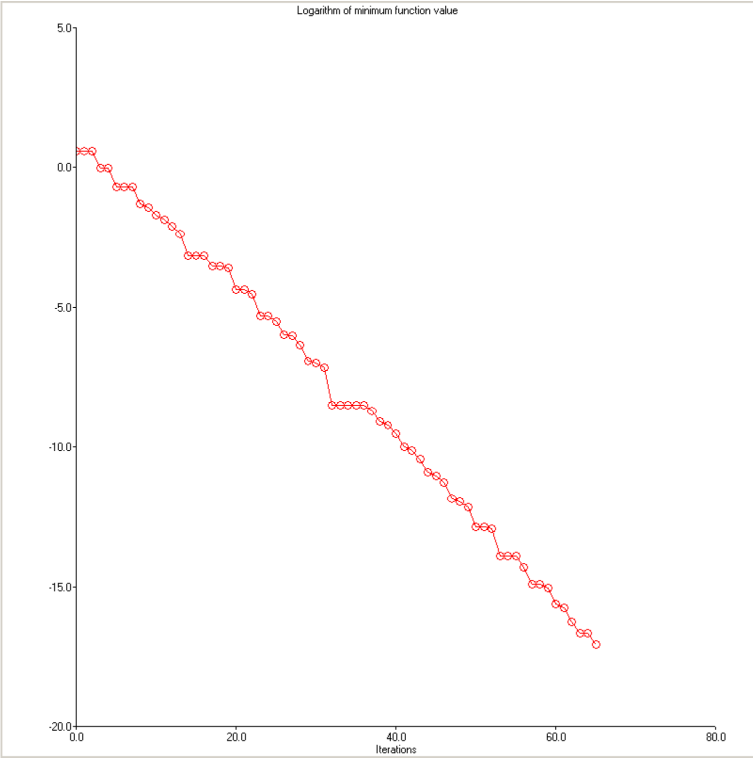

Fig. 5.16 : Nelder-Mead numerical experiment – history of logarithm of function

that the function value at optimum is very accurate ($f(x^\star) \approx 1.e-17$ compared to Spendley's et al. $f(x^\star) \approx 0.08$).

In figure 5.18, we analyse the behaviour of the method with respect to scaling. We check that the method behave very smoothly, with a very small number of additionnal function evaluations when the scaling deteriorates. This shows how much the Nelder-Mead algorithms improves over the Spendley et al. method.

5.4.2 Sensitivity to dimension

In this section, we try to reproduce the result presented by Han and Neumann [11], which shows that the convergence rate of the Nelder-Mead algorithms rapidly deteriorates when the number of variables increases. The function we try to minimize is the following quadratic in n-dimensions

$$f(\mathbf{x}) = \sum_{i=1,n} x_i^2. \tag{5.31}$$

The initial simplex is computed from the coordinate axis and the unit length. The initial guess is at 0 so that the first vertex is the origin ; this vertex is never updated during the iterations.

The figure 5.21 presents the results of this experiment for $n = 1, 19$.

During the iterations, no shrink steps are performed. The algorithm performs reflections, inside and outside contractions. The figure 5.19 shows the detailed sequence of iterations for

	Nelder-Mead	Spendley et al.
Iterations	83	340
Function Evaluations	161	400
a	100.0	-
x_0	$(10.0, 10.0)$	-
Relative tolerance on simplex size	-	
Exact $x^\star$	$(0., 0.)$	-
Computed $x^\star$	$(2.e-10, -3.e-9)$	$(0.001, 0.2)$
Computed $f(x^\star)$	$1.e-17$	0.08

Fig. 5.17 : 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	139	$8.0e-18$	$(2.e-9-1.e-9)$
10.0	151	$7.0e-17$	$(5.e-102.e-9)$
100.0	161	$1.0e-17$	$(2.e-10-3.e-9)$
1000.0	165	$1.0e-17$	$(-1.e-0109.e-10)$
10000.0	167	$3.0e-17$	$(5.0e-11, -1.0e-10)$

Fig. 5.18 : Numerical experiment with Spendley's et al. method on a badly scaled quadratic function

$n = 10$. One can see that there is no general pattern for the iterations. One can check, however, that there are never no more than n consecutive reflection steps, which is as expected. After one or more contractions, the reflection steps move the worst vertices toward better function values. But there are only $n+1$ vertices so that the n worst vertices are moved in at most n reflection steps.

```
I I I I I I I I I I I I I I I I I I I I R R R R R R R R R R R R R I O
R R R R R R R R R R R R R I R I I R I R O I I I I R I R I I R O I R R
R R I R I R I R R R R R R R I R R R R I R I I R I R I I I R R I I I R
R R I R R I R R R R R R I R I R R R R R I R R O R R I O I O R R R R I
I I O R I R R R R R R I I I R R I I R R R O R I I R R R I R I I O I R
I R R O I I R R R R I R R O I R R O R I R I R I R R I R I R R R I I I
I I O R R R R I I I R R R I I I R R R I I I I R R R R I I R R R R I R
R R I O I R R I I R R R R O I R I I R R R R R R O R R R O I R R I I I
I O R I I I R I I I I R R I I R R I R R R R R R I R R I I R R O R I I
O R I R O O R O I I R I I I R I I R R R R R R R R R R R R I R R O I R
I O I R I I I I R I I R I I R I R O R I O R I R I R R R O R I R R R I
I R I R R R I R I R R R R I I R R I R R R R I I R R R R I I R I R I I
O I R I I R R R R R R R R I O I R R I I I R I R I I I I R R R R I R R
I R I R R R R I I R R R I I I I R I I I I R I R R I R I O R R R I R I
O I R R I I R R I R R R R O R R R R I O R R R I R I I I I R I R R R R
R I I R I I R R R R R O R R R I R R R R I R I R R I R I I R R I I R R
I I I I R R R R R R R R R I R R O R R R R R O I I I I R I I R O I I R
R R R R I I I R R
```

Fig. 5.19 : Numerical experiment with Nelder-Mead method on a generalized quadratic function - steps of the algorithm : I = inside contraction, O = outside contraction, R = reflection, S = shrink

The figure 5.20 presents the number and the kind of steps performed during the iterations for $n = 1, 19$. It appears that the number of shrink steps and expansion steps is zero, as expected. More interesting is that the number of reflection is larger than the number of inside contraction when n is large. The number of outside contraction is allways the smallest in this case.

One can check that the number of function evaluations increases approximately linearily with the dimension of the problem in figure 5.22. A rough rule of thumb is that, for $n = 1, 19$, the

n	# Reflections	# Expansion	# Inside Contractions	# Outside Contractions	#Shrink
1	0	0	27	0	0
2	0	0	5	49	0
3	54	0	45	36	0
4	93	0	74	34	0
5	123	0	101	33	0
6	170	0	122	41	0
7	202	0	155	35	0
8	240	0	178	41	0
9	267	0	205	40	0
10	332	0	234	38	0
11	381	0	267	36	0
12	476	0	299	32	0
13	473	0	316	42	0
14	545	0	332	55	0
15	577	0	372	41	0
16	635	0	396	46	0
17	683	0	419	52	0
18	756	0	445	55	0
19	767	0	480	48	0

Fig. 5.20 : Numerical experiment with Nelder-Mead method on a generalized quadratic function – number and kinds of steps performed

number of function evaluations is equal to $100n$.

n	Function evaluations	Iterations	$\rho(S_0, n)$
1	56	28	0.5125321059829373
2	111	55	0.71491052830553248
3	220	136	0.87286283470760984
4	314	202	0.91247307800713973
5	397	258	0.93107793607270162
6	503	334	0.94628781077508028
7	590	393	0.95404424343636474
8	687	460	0.96063768057900478
9	767	513	0.96471820169933631
10	887	605	0.97000569588245511
11	999	685	0.97343652480535203
12	1151	808	0.97745310525741003
13	1203	832	0.97803465666405531
14	1334	933	0.98042500139065414
15	1419	991	0.98154526298964495
16	1536	1078	0.98305435726547608
17	1643	1155	0.98416149958157839
18	1775	1257	0.98544909490809807
19	1843	1296	0.98584701106083183

Fig. 5.21 : Numerical experiment with Nelder-Mead method on a generalized quadratic function

The figure 5.23 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.

5.4.3 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 [27].

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,

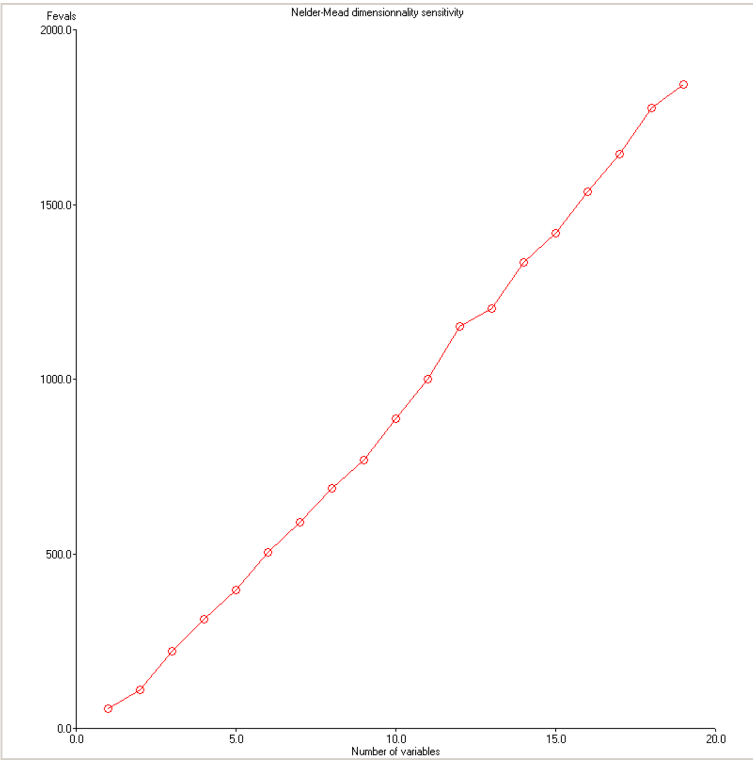

Fig. 5.22 : Nelder-Mead numerical experiment – number of function evaluations depending on the number of variables

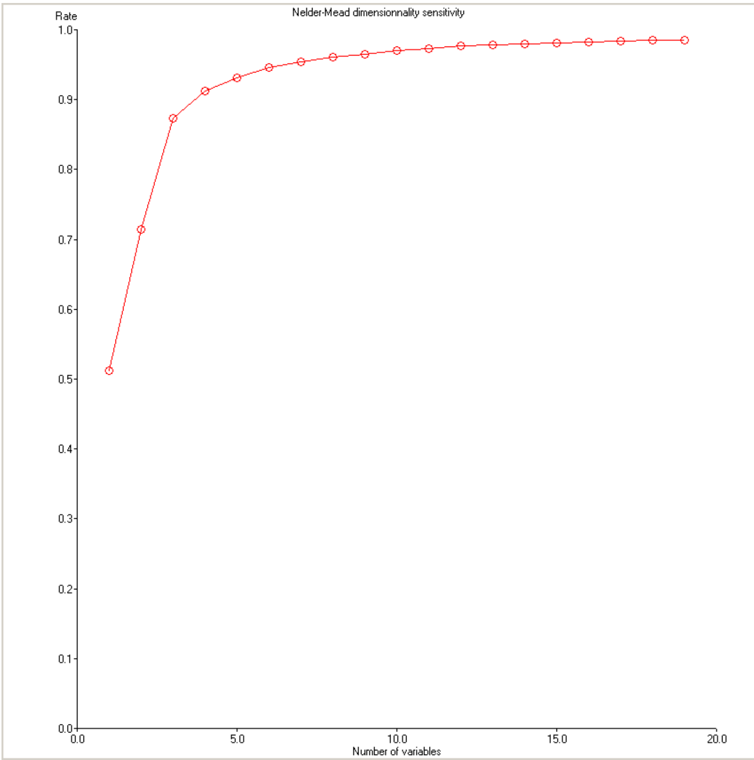

Fig. 5.23 : Nelder-Mead numerical experiment – rate of convergence depending on the number of variables

- 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.

The following tests are presented by O'Neill :

- Rosenbrock's parabolic valley [35]

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{5.32}$$

  with starting point $(x_1, x_2) = (-1.2, 1.0)$

- Powell's quartic function [32]

$$f(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \tag{5.33}$$

  with starting point $(x_1, x_2, x_3, x_4) = (3, -1, 0, 1)$

- Fletcher and Powell's helical valley [8]

$$f(x_1, x_2, x_3) = 100\left(x_3 + 10\theta(x_1, x_2)\right)^2 + \left(\sqrt{x_1^2 + x_2^2} - 1\right)^2 + x_3^2 \tag{5.34}$$

  where

$$\begin{aligned} 2\pi\theta(x_1, x_2) &= \arctan(x_2, x_1), x_1 > 0 && (5.35)\\ &= \pi + \arctan(x_2, x_1), x_1 < 0 && (5.36)\\ & && (5.37) \end{aligned}$$

  with starting point $(x_1, x_2, x_3) = (-1, 0, 0)$. 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

$$f(x_1, \ldots, x_{10}) = \sum_{i=1,10} x_i^4 \tag{5.38}$$

  with starting point $(x_1, \ldots, x_{10}) = (1, \ldots, 1)$

The parameters are set to

- $REQMIN = 10^{-16}$, the absolute tolerance on the variance of the function values in the simplex,

- $STEP = 1.0$, the absolute side length of the initial simplex,
- $ICOUNT$, the maximum number of function evaluations.

The table 5.24 presents the results which were computed by O'Neill compared with our software. For most experiments, the results are very close in terms of number of function evaluations. The problem #4 exhibits a completely different behaviour than the results presented by O'Neill. For us, the maximum number of function evaluations is reached (i.e. 1000 function evaluations), whereas for O'Neill, the algorithm is restarted and gives the result with 474 function evaluations. We did not find any explanation for this behaviour. 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	No. of Restarts	Function Value	Iterations	CPU Time
O'Neill	1	148	0	3.19 e-9	?	?
Baudin	1	149	0	1.15 e-7	79	0.238579
O'Neill	2	209	0	7.35 e-8	?	?
Baudin	2	224	0	1.07 e-8	126	0.447958
O'Neill	3	250	0	5.29 e-9	?	?
Baudin	3	255	0	4.56 e-8	137	0.627493
O'Neill	4	474	1	3.80 e-7	?	?
Baudin	4	999	0	5.91 e-9	676	-

Fig. 5.24 : Numerical experiment with Nelder-Mead method on O'Neill test cases - O'Neill results and our results

5.4.4 Convergence to a non stationnary point

In this section, we analyse the Mc Kinnon counter example from [18]. We show the behavior of the Nelder-Mead simplex method for a family of examples which cause the method to converge to a nonstationnary point.

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

$$f(0) < f(v^{(n+1)}) < f(v^{(n)}). \tag{5.39}$$

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

$$r^{(n)} = \overline{v} + \rho(\overline{v} - v^{(n)}) = v^{(n+1)} - v^{(n)} \tag{5.40}$$

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

$$v^{(n+2)} = \overline{v} - \gamma(\overline{v} - v^{(n)}) = \frac{1}{4}v^{(n+1)} - \frac{1}{2}v^{(n)} \tag{5.41}$$

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

$$4v^{(n+2)} - v^{(n+1)} + 2v^{(n)} = 0 \tag{5.42}$$

Their general solutions are of the form

$$v^{(n)} = \lambda_1^k a_1 + \lambda_2^k a_2 \tag{5.43}$$

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

$$4\lambda^2 - \lambda + 2\lambda = 0 \tag{5.44}$$

and has the roots

$$\lambda_1 = \frac{1+\sqrt{33}}{8} \approx 0.84307, \qquad \lambda_2 = \frac{1-\sqrt{33}}{8} \approx -0.59307 \tag{5.45}$$

After Mc Kinnon has presented the computation of the roots of the caracteristic 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 $v^{(0)} = (1,1)$ and $v^{(1)} = (\lambda_1, \lambda_2)$ and 0. If follows that the particular solution for these initial conditions is $v^{(n)} = (\lambda_1^n, \lambda_2^n)$.

Consider the function $f(x,y)$ given by

$$\begin{aligned} f(x,y) &= \theta\phi|x|^\tau + y + y^2, \qquad x \leq 0, & (5.46)\\ &= \theta x^\tau + y + y^2, \qquad x \geq 0. & (5.47) \end{aligned}$$

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 continous third derivatives if $\tau > 3$.

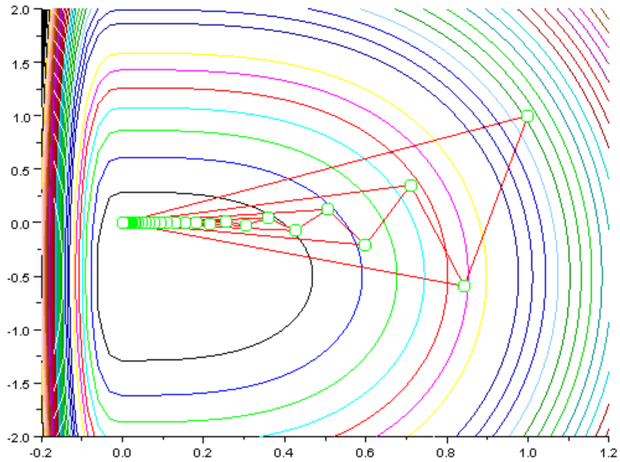

Fig. 5.25 : Nelder-Mead numerical experiment – Mc Kinnon example for convergence toward a non stationnary point

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

$$
\begin{aligned}
f(x, y) &= -2400x^3 + y + y^2, \qquad x \leq 0, && (5.48)\\
&= 6x^3 + y + y^2, \qquad x \geq 0. && (5.49)
\end{aligned}
$$

The figure 5.25 shows the contour plot of this function and the first steps of the Nelder-Mead method.

5.4.5 Han counter examples

In his Phd thesis [10], 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

$$
f(x, y) = x^2 + y(y + 2)(y - 0.5)(y - 2) \tag{5.50}
$$

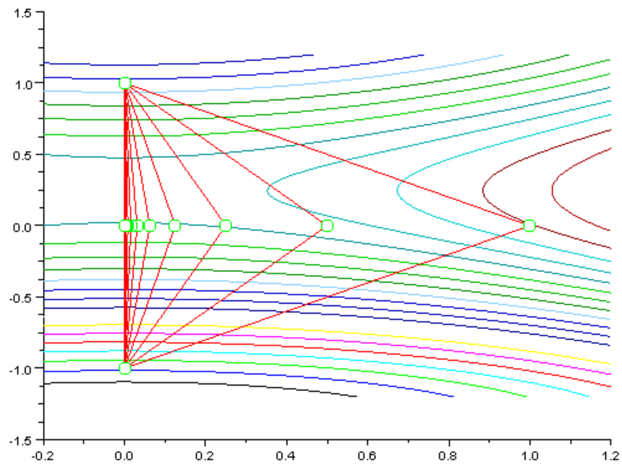

Fig. 5.26 : Nelder-Mead numerical experiment – Han example #1 for convergence toward a non stationnary point

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 prooves that the Nelder-Mead algorithm generates a sequence of simplices $S_k = [(0., -1), (0, 1), (\frac{1}{2^k}, 0)]$.

The figure 5.26 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

$$f(x, y) = x^2 + \rho(y) \tag{5.51}$$

where ρ is a continuous convex function with bounded level sets which satisfies

$$\begin{aligned} \rho(y) &= 0, \quad \text{if} \quad |y| \leq 1, & (5.52)\\ &\geq 0, \quad \text{if} \quad |y| > 1. & (5.53) \end{aligned}$$

The example given by Han for such a ρ function is

$$\begin{aligned} \rho(y) &= 0, \quad \text{if} \quad |y| \leq 1, & (5.54)\\ &= y - 1, \quad \text{if} \quad y > 1, & (5.55)\\ &= -y - 1, \quad \text{if} \quad y < -1. & (5.56) \end{aligned}$$

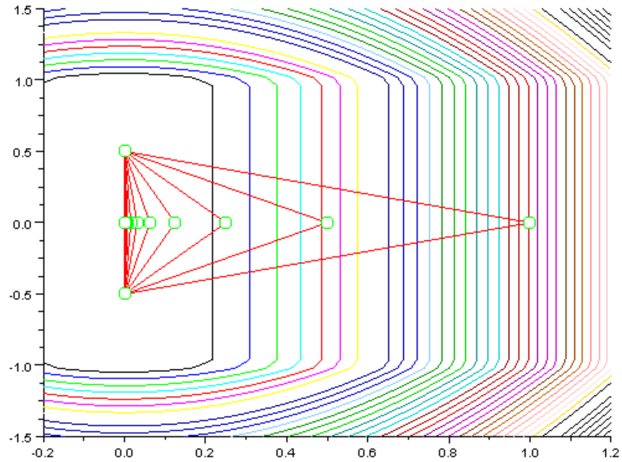

Fig. 5.27 : Nelder-Mead numerical experiment – Han example #2 for convergence toward a non stationnary point

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 $S_k = [(0., 1/2), (0, -1/2), (\frac{1}{2^k}, 0)]$.

The figure 5.27 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.

5.4.6 Torczon's numerical experiments

In her Phd Thesis [39], Virginia Torczon presents the multi-directionnal direct search algorithm. In order to analyse 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 [19], published in the ACM by MorÃl', 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 90Âř. 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

$$f(x) = \sum_{i=1,m} f_i(x)^2, \tag{5.57}$$

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

$$\frac{1}{\Delta} \max_{i=1,n} \|v_i^k - v_0^k\| \leq \epsilon, \tag{5.58}$$

where $\Delta = \max(1, \|v_0^k\|)$. Decreasing the value of ϵ allows to get smaller simplex sizes.

Penalty #1

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

$$f_i(x) = \sqrt{1.e-5}(x_i - 1), \qquad i = 1, n \tag{5.59}$$
$$f_{n+1} = -\frac{1}{4} + \sum_{j=1,n} x_j^2. \tag{5.60}$$

The initial guess is given by $x_0 = (x_{0,j})_{j=1,n}$ and $x_{0,j} = j$ for $j = 1, n$.

The problem given by MorÃl', Garbow and Hillstrom in [19] is associated with the size $n = 4$. The value of the cost function at the initial guess $x_0 = (1, 2, 3, 4)$ is $f(x_0) = 885.063$. The value of the function at the optimum is given in [19] as $f(x^\star) = 2.24997d - 5$.

Torzcon shows an experiment with the Penalty #1 test case and $n = 8$. For this particular case, the initial function value is $f(x_0) = 4.151406e4$. The figure 5.28 presents the results of these experiments. The number of function evaluations is not the same so that we can conclude that the algorithm may be different variants of the Nelder-Mead algorithms. We were not able to explain why the number of function evaluations is so different.

Author	Step Tolerance	$f(v_0^star)$	Function Evaluations	Angle (,°)
Torzcon	1.e-1	7.0355e-5	1605	89.396677792198
Baudin	1.e-1	8.2272e-5	530	87.7654
Torzcon	1.e-2	6.2912e-5	1605	89.935373548613
Baudin	1.e-2	7.4854e-5	1873	89.9253
Torzcon	1.e-3	6.2912e-5	3600	89.994626919197
Baudin	1.e-3	7.4815e-5	2135	90.0001
Torzcon	1.e-4	6.2912e-5	3670	89.999288284747
Baudin	1.e-4	7.481546e-5	2196	89.9991
Torzcon	1.e-5	6.2912e-5	3750	89.999931862232
Baudin	1.e-5	7.427212e-5	4626	89.999990

Fig. 5.28 : Numerical experiment with Nelder-Mead method on Torczon test cases - Torczon results and our results

The figure 5.29 presents the angle between the gradient of the function $-g_k$ and the search direction $x_c - x_h$, where x_c is the centroid of the best points and x_h is the worst (or high) vertex.

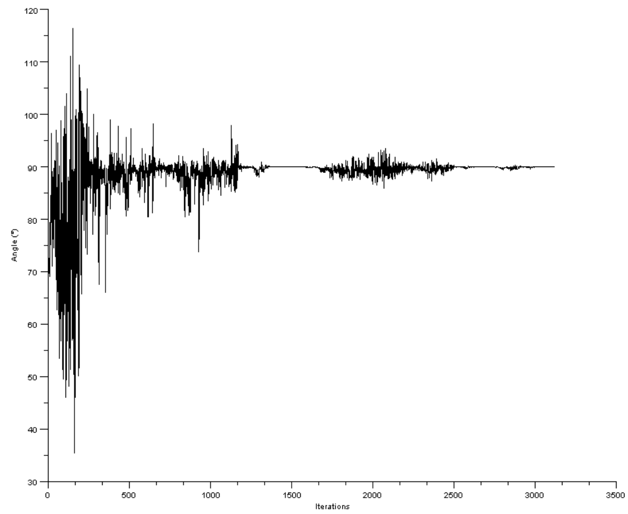

Fig. 5.29 : Nelder-Mead numerical experiment – Penalty #1 function – One can see that the angle between the gradient and the search direction is very close to $90°$, especially for large number of iterations.

The numerical experiment shows that the conditioning of the matrix of simplex direction has an increasing condition number. This corresponds to the fact that the simplex is increasingly distorted.

5.5 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. Nevertheless, the behaviour 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 $100n$.

Chapter 6

Conclusion

That tool might be extended in future releases so that it provides the following features :

- Kelley restart based on simplex gradient [9],
- C-based implementation (a prototype is provided in appendix B),
- parallel implementation of the DIRECT algorithm,
- implementation of the Hook-Jeeves and Multidimensional Search methods [9]
- parallel implementation of the Nelder-Mead algorithm. See for example [21]. ?This paper generalizes the widely used Nelder and Mead (Comput J 7:308?313, 1965) simplex algorithm to parallel processors. Unlike most previous parallelization methods, which are based on parallelizing the tasks required to compute a specific objective function given a vector of parameters, our parallel simplex algorithm uses parallelization at the parameter level. Our parallel simplex algorithm assigns to each processor a separate vector of parameters corresponding to a point on a simplex. The processors then conduct the simplex search steps for an improved point, communicate the results, and a new simplex is formed. The advantage of this method is that our algorithm is generic and can be applied, without re-writing computer code, to any optimization problem which the non-parallel Nelder?Mead is applicable. The method is also easily scalable to any degree of parallelization up to the number of parameters. In a series of Monte Carlo experiments, we show that this parallel simplex method yields computational savings in some experiments up to three times the number of processors.?

Chapter 7

Acknowledgments

I would like to thank Vincent Couvert, the team manager for Scilab releases, for his support during the development of this software. I would like to thank Serge Steer, INRIA researcher, for his comments and the discussions on this subject. Professor Han, Associate Professor of Mathematics in the University of Michigan-Flint University was kind enough to send me a copy of his Phd and I would like to thank him for that.

Appendix A

Nelder-Mead bibliography

In this section, we present a brief overview of selected papers, sorted in chronological order, which deals with the Nelder-Mead algorithm

A.1 Spendley, Hext, Himsworth, 1962

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", Spendley W., Hext G. R. and Himsworth F. R., American Statistical Association and American Society for Quality, 1962

This article [37] presents an algorithm for unconstrained optimization in which a simplex is used. The simplex has a fixed, regular (i.e. all lengths are equal), shape and is made of n+1 vertices (where n is the number of parameters to optimize). The algorithm is based on the reflection of the simplex with respect to the centroid of better vertices. One can add a shrink step so that the simplex size can converge to zero. Because the simplex shape cannot change, the convergence rate may be very slow if the eigenvalues of the hessian matrix have very different magnitude.

A.2 Nelder, Mead, 1965

"A Simplex Method for Function Minimization", Nelder J. A. and Mead R., The Computer Journal, 1965

This article [25] presents the Nelder-Mead unconstrained optimization algorithm. It is based on a simplex made of n+1 vertices and is a modification of the Spendley's et al algorithm. It includes features which enables the simplex to adapt to the local landscape of the cost function. The additional steps are expansion, inside contraction and outside contraction. The stopping criterion is based on the standard deviation of the function value on the simplex.

The convergence of the algorithm is better than Spendley's et al. The method is compared against Powell's free-derivative method (1964) with comparable behaviour. The algorithm is "greedy" in the sense that the expansion point is kept if it improves the best function value in the current simplex. Most Nelder-Mead variants which have been analyzed after are keeping the expansion point only if it improves over the reflection point.

A.3 Box, 1965

"A New Method of Constrained Optimization and a Comparison With Other Methods", M. J. Box, The Computer Journal 1965 8(1):42-52, 1965, British Computer Society

In this paper [4], Box presents a modification of the NM algorithm which takes into accounts for bound constraints and non-linear constraints. This variant is called the Complex method. The method expects that the initial guess satisfies the nonlinear constraints. The nonlinear constraints are supposed to define a convex set. The algorithm ensures that the simplex evolves in the feasible space.

The method to take into account for the bound constraints is based on projection of the parameters inside the bounded domain. If some nonlinear constraint is not satisfied, the trial point is moved halfway toward the centroid of the remaining points (which are all satisfying the nonlinear constraints).

The simplex may collapse into a subspace if a projection occurs. To circumvent this problem, k>=n+1 vertices are used instead of the original n+1 vertices. A typical value of k is k=2n. The initial simplex is computed with a random number generator, which takes into account for the bounds on the parameters. To take into account for the nonlinear constraints, each vertex of the initial simplex is moved halfway toward the centroid of the points satisfying the constraints (in which the initial guess already is).

A.4 Guin, 1968

"Discussion and correspondence: modification of the complex method of constrained optimization", J. A. Guin, The Computer Journal, 1968

In this article [9], Guin suggest 3 rules to improve the practical convergence properties of Box's complex method. These suggestions include the use of the next-to-worst point when the worst point does not produce an improvement of the function value. The second suggestion is to project the points strictly into the bounds, instead of projecting inside the bounds. The third suggestion is related to the failure of the method when the centroid is no feasible. In that case, Guin suggest to restrict the optimization in the subspace defined by the best vertex and the centroid.

A.5 O'Neill, 1971

"Algorithm AS47 - Function minimization using a simplex procedure", R. O'Neill, 1971, Applied Statistics

In this paper [27], R. O'Neill presents a fortran 77 implementation of the Nelder-Mead algorithm. The initial simplex is computed axis-by-axis, given the initial guess and a vector of step lengths. A factorial test is used to check if the computed optimum point is a local minimum.

A.5.1 Parkinson and Hutchinson, 1972

In [29], "An investigation into the efficiency of variants on the simplex method", Parkinson and Hutchinson explored several ways of improvement. First, they investigate the sensitivity of the algorithm to the initial simplex. Two parameters were investigated, i.e. the initial length and the orientation of the simplex. An automatic setting for the orientation, though very desirable, is not easy to design. Parkinson and Hutchinson tried to automatically compute the scale of the initial simplex by two methods, based on a "line search" and on a local "steepest descent". Their second investigation adds a new step to the algorithm, the unlimited expansion. After a sucessful expansion, the algorithm tries to produce an expansion point by taking the largest possible number of expansion steps. After an unlimited expansion steps is performed, the simplex is translated, so that excessive modification of the scale and shape is avoided. Combined and tested against low dimension problems, the modified algorithm, named PHS, provides typical gains of 20function evaluations.

A.6 Richardson and Kuester, 1973

"Algorithm 454: the complex method for constrained optimization", Richardson Joel A. and Kuester J. L., Commun. ACM, 1973

In this paper [34], Richardson and Kuester shows a fortran 77 implementation of Box's complex optimization method. The paper clarifies several specific points from Box's original paper while remaining very close to it. Three test problems are presented with the specific algoritmic settings (such as the number of vertices for example) and number of iterations.

A.7 Shere, 1973

"Remark on algorithm 454 : The complex method for constrained optimization", Shere Kenneth D., Commun. ACM, 1974

In this article [36], Shere presents two counterexamples where the algorithm 454, implemented by Richardson and Kuester produces an infinite loop. "This happens whenever the corrected point, the centroid of the remaining complex points, and every point on the line segment joining these two points all have functional values lower than the functional values at each of the remaining complex points.

A.8 Subrahmanyam, 1989

"An extension of the simplex method to constrained nonlinear optimization", M. B. Subrahmanyam, Journal of Optimization Theory and Applications, 1989

In this article [38], the simplex algorithm of Nelder and Mead is extended to handle nonlinear optimization problems with constraints. To prevent the simplex from collapsing into a subspace near the constraints, a delayed reflection is introduced for those points moving into the infeasible region. Numerical experience indicates that the proposed algorithm yields good results in the presence of both inequality and equality constraints, even when the constraint region is narrow.

If a vertex becomes infeasible, we do not increase the value at this vertex until the next iteration is completed. Thus, the next iteration is accomplished using the actual value of the function at the infeasible point. At the end of the iteration, in case the previous vertex is not the worst vertex, it is assigned a high value, so that it then becomes a candidate for reflection during the next iteration.

The paper presents numerical experiments which are associated with thousands of calls to the cost function. This may be related with the chosen reflection factor equal to 0.95, which probably cause a large number of reflections until the simplex can finally satisfy the constraints.

A.9 Numerical Recipes in C, 1992

"Numerical Recipes in C, Second Edition", W. H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, 1992

In this book [33], an ANSI C implementation of the Nelder-Mead algorithm is given. The initial simplex is based on the axis. The termination criterion is based on the relative difference of the function value of the best and worst vertices in the simplex.

A.10 Lagarias, Reeds, Wright, Wright, 1998

"Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions", Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright and Paul E. Wright, SIAM Journal on Optimization, 1998

This paper [16] presents convergence properties of the Nelder-Mead algorithm applied to stricly convex functions in dimensions 1 and 2. Proofs are given to a minimizer in dimension 1, and various limited convergence results for dimension 2.

A.11 Mc Kinnon, 1998

"Convergence of the Nelder–Mead Simplex Method to a Nonstationary Point", SIAM J. on Optimization, K. I. M. McKinnon, 1998

In this article [18], Mc Kinnon analyses the behavior of the Nelder-Mead simplex method for a family of examples which cause the method to converge to a nonstationnary point. All the examples use continuous functions of two variables. The family of functions contains strictly convex functions with up to three continuous derivatives. In all the examples, the method repeatedly applies the inside contraction step with the best vertex remaining fixed. The simplices tend to a straight line which is orthogonal to the steepest descent direction. It is shown that this behavior cannot occur for functions with more than three continuous derivatives.

A.12 Kelley, 1999

"Detection and Remediation of Stagnation in the Nelder–Mead Algorithm Using a Sufficient Decrease Condition", SIAM J. on Optimization, Kelley, C. T., 1999

In this article [14], Kelley presents a test for sufficient decrease which, if passed for the entire iteration, will guarantee convergence of the Nelder-Mead iteration to a stationary point if the objective function is smooth. Failure of this condition is an indicator of potential stagnation. As a remedy, Kelley propose to restart the algorithm with an oriented simplex, smaller than the previously optimum simplex, but with a better shape and which approximates the steepest descent step from the current best point. The method is experimented against Mc Kinnon test function and allow to converge to the optimum, where the original Nelder -Mead algorithm was converging to a non-stationary point. Although the oriented simplex works well in practice, other strategies may be chosen with similar results, such as a simplex based on axis, a regular simplex (like Spendley's) or a simplex based on the variable magnitude (like Pfeffer's suggestion in Matlab's fminsearch). The paper also shows one convergence theorem which prove that if the sufficient decrease condition is satisfied and if the product of the condition of the simplex by the simplex size converge to zero, therefore, with additional assumptions on the cost function and the sequence of simplices, any accumulation point of the simplices is a critical point of f.

The same ideas are presented in the book [15].

A.12.1 Han, 2000

In his Phd thesis [10], Lixing Han analyse the properties of the Nelder-Mead algorithm. Han present two examples in which the Nelder-Mead simplex method does not converge to a single point. The first example is a nonconvex function with bounded level sets and it exhibits similar nonconvergence properties with the Mc Kinnon counterexample $f(\xi_1, \xi_2) = \xi_1^2 - \xi_2(\xi_2 - 2)$. The second example is a convex function with bounded level sets, for which the Nelder-Mead simplices converge to a degenerate simplex, but not to a single point. These nonconvergent examples support the observations by some practitionners that in the Nelder-Mead simplices may collapse into a degenerate simplex and therefore support the use of a restart strategy. Han also investigates the effect of the dimensionality of the Nelder-Mead method. It is shown that the

Nelder-Mead simplex method becomes less efficient as the dimension increases. Specifically, Han consider the quadratic function $\xi_1^2 + \ldots + \xi_1^n$ and shows that the Nelder-Mead method becomes less efficient as the dimension increases. The considered example offers insight into understanding the effect of dimensionnality on the Nelder-Mead method. Given all the known failures and inefficiencies of the Nelder-Mead method, a very interesting question is why it is so popular in practice. Han present numerical results of the Nelder-Mead method on the standard collection of MorÃl'-Garbow-Hillstrom with dimensions $n \leq 6$. Han compare the Nelder-Mead method with a finite difference BFGS method and a finite difference steepest descent method. The numerical results show that the Nelder-Mead method is much more efficient than the finite difference steepest descent method for the problems he tested with dimensions $n \leq 6$. It is also often comparable with the finite difference BFGS method, which is believed to be the best derivative-free method. Some of these results are reproduced in [11] by Han and Neumann, "Effect of dimensionality on the NelderâĂŞMead simplex method" and in [12], "On the roots of certain polynomials arising from the analysis of the Nelder-Mead simplex method".

A.13 Nazareth, Tseng, 2001

"Gilding the Lily: A Variant of the Nelder-Mead Algorithm Based on Golden-Section Search" Computational Optimization and Applications, 2001, Larry Nazareth and Paul Tseng

The article [24] propose a variant of the Nelder-Mead algorithm derived from a reinterpretation of univariate golden-section direct search. In the univariate case, convergence of the variant can be analyzed analogously to golden-section search.

The idea is based on a particular choice of the reflection, expansion, inside and outside contraction parameters, based on the golden ratio. This variant of the Nelder-Mead algorithm is called Nelder-Mead-Golden- Ratio, or NM-GS. In one dimension, the authors exploit the connection with golden-search method and allows to prove a convergence theorem on unimodal univariate functions. This is marked contrast to the approach taken by Lagarias et al. where considerable effort is expended to show convergence of the original NM algorithm on strictly convex univariate functions. With the NM-GS variant, one obtain convergence in the univariate case (using a relatively simple proof) on the broader class of unimodal functions.

In the multivariate case, the authors modify the variant by replacing strict descent with fortified descent and maintaining the interior angles of the simplex bounded away from zero. Convergence of the modified v ariant can be analyzed by applying results for a fortified- descent simplicial search method. Some numerical experience with the variant is reported.

A.14 Perry, Perry, 2001

"A New Method For Numerical Constrained Optimization" by Ronald N. Perry, Ronald N. Perry, March 2001

In this report [30], we propose a new method for constraint handling that can be applied to established optimization algorithms and which significantly improves their ability to traverse through constrained space. To make the presentation concrete, we apply the new constraint method to the Nelder and Mead polytope algorithm. The resulting technique, called SPIDER, has shown great initial promise for solving difficult (e.g., nonlinear, nondifferentiable, noisy) constrained problems.

In the new method, constraints are partitioned into multiple levels. A constrained performance, independent of the objective function, is defined for each level. A set of rules, based on these partitioned performances, specify the ordering and movement of vertices as they straddle constraint boundaries; these rules [...] have been shown to significantly aid motion along constraints toward an optimum. Note that the new approach uses not penalty function and thus does not warp the performance surface, thereby avoiding the possible ill-conditioning of the objective function typical in penalty methods.

No numerical experiment is presented.

A.15 Andersson, 2001

"Multiobjective Optimization in Engineering Design - Application to fluid Power Systems" Johan Andersson, 2001

This PhD thesis [2] gives a brief overview of the Complex method by Box in section 5.1.

A.15.1 Peters, Bolte, Marschner, Nüssen and Laur, 2002

In [31], "Enhanced Optimization Algorithms for the Development of Microsystems", the authors combine radial basis function interpolation methods with the complex algorithm by Box. Interpolation with radial basis functions is a linear approach in which the model function f is generated via the weighted sum of the basis functions $\Phi_i(r)$. The parameter r describes the distance of the current point from the center x_i of the ith basis function. It is calculated via the euclidean norm. It is named ComplInt strategy. The name stands for Complex in combination with interpolation. The Complex strategy due to Box is very well suited for the combination with radial basis function interpolation for it belongs to the polyhedron strategies. The authors presents a test performed on a pratical application, which leaded them to the following comment : "The best result achieved with the ComplInt strategy is not only around 10the Complex strategy due to Box, the ComplInt also converges much faster than the Complex does: while the Complex strategy needs an average of 7506, the ComplInt only calls for an average of 2728 quality function evaluations."

A.16 Han, Neumann, 2006

"Effect of dimensionality on the Nelder-Mead simplex method", L. Han and M. Neumann (2006), In this article [11], the effect of dimensionality on the Nelder-Mead algorithm is investigated. It is shown that by using the quadratic function $f(x) = x^T * x$, the Nelder-Mead simplex method deteriorates as the dimension increases. More precisely, in dimension 1, with the quadratic function $f(x) = x^2$ and a particular choice of the initial simplex, applies inside contraction step repeatedly and the convergence rate (as the ratio between the length of the simplex at two consecutive steps) is 1/2. In dimension 2, with a particular initial simplex, the NM algorithm applies outside contraction step repeatedly and the convergence rate is $\sqrt(2)/2$.

For n>=3, a numerical experiment is performed on the quadratic function with the fminsearch algorithm from Matlab. It is shown that the original NM algorithm has a convergence rate which is converging towards 1 when n increases. For n=32, the rate of convergence is 0.9912.

A.17 Singer, Nelder, 2008

http://www.scholarpedia.org/article/Nelder-Mead_algorithm Singer and Nelder

This article is a complete review of the Nelder-Mead algorithm. Restarting the algorithm is adviced when a premature termination occurs.

Appendix B

Implementations of the Nelder-Mead algorithm

In the following sections, we analyse the various implementations of the Nelder-Mead algorithm. We analyse the Matlab implementation provided by the *fminsearch* command. We analyse the matlab algorithm provided by C.T. Kelley and the Scilab port by Y. Collette. We present the Numerical Recipes implementations. We analyse the O'Neill fortran 77 implementation "AS47". The Burkardt implementation is also covered. The implementation provided in the NAG collection is detailed. The Nelder-Mead algorithm from the Gnu Scientific Library is analysed.

B.1 Matlab : fminsearch

The Matlab command fminsearch implements the Nelder-Mead algorithm [17]. It provides features such as

- maximum number of function evaluations,
- maximum number of iterations,
- termination tolerance on the function value,
- termination tolerance on x,
- output command to display the progress of the algorithm.

B.2 Kelley and the Nelder-Mead algorithm

C.T. Kelley has written a book [15] on optimization method and devotes a complete chapter to direct search algorithms, especially the Nelder-Mead algorithm. Kelley provides in [13] the Matlab implementation of the Nelder-Mead algorithm. That implementation uses the restart strategy that Kelley has published in [14] and which improves the possible stagnation of the algorithm on non local optimization points. No tests are provided.

The following is extracted from the README provided with these algorithms.

```
These files are current as of December 9, 1998.

-----------------

MATLAB/FORTRAN software for Iterative Methods for Optimization

by C. T. Kelley

These M-files are implementations of the algorithms from the book
"Iterative Methods for Optimization", to be published by SIAM,
by C. T. Kelley. The book, which describes the algorithms, is available
from SIAM (service@siam.org). These files can be modified for non-commercial
purposes provided that the authors:

C. T. Kelley for all MATLAB codes,
P. Gilmore and T. D. Choi for iffco.f
J. M. Gablonsky for DIRECT

are acknowledged and clear comment lines are inserted
that the code has been changed. The authors assume no no responsibility
for any errors that may exist in these routines.

Questions, comments, and bug reports should be sent to

Professor C. T. Kelley
Department of Mathematics, Box 8205
North Carolina State University
Raleigh, NC 27695-8205

(919) 515-7163
(919) 515-3798 (FAX)

Tim_Kelley@ncsu.edu
```

From Scilab's point of view, that ?licence? is a problem since it prevents the use of the source for commercial purposes.

B.3 Nelder-Mead Scilab Toolbox : Lolimot

The Lolimot project by Yann Collette provide two Scilab-based Nelder- Mead implementations [5]. The first implementation is a Scilab port of the Kelley script. The licence problem is therefore not solved by this script. The second implementation [6] implements the restart strategy by Kelley. No tests are provided.

B.4 Numerical Recipes

The Numerical Recipes [33] provides the C source code of an implementation of the Nelder-Mead algorithm. Of course, this is a copyrighted material which cannot be included in Scilab.

B.5 NASHLIB : A19

Nashlib is a collection of Fortran subprograms from "Compact Numerical Methods for Computers; Linear Algebra and Function Minimisation, " by J.C. Nash. The subprograms are written without many of the extra features usually associated with commercial mathematical software, such as extensive error checking, and are most useful for those applications where small program size is particularly important. The license is public domain.

Nahslib includes one implementation of the Nelder-Mead algorithm [22], [23]. It is written in fortran 77. The coding style is "goto"-based and may not be easy to maintain.

B.6 O'Neill implementations

The paper [27] by R. O'Neil in the journal of Applied Statistics presents a fortran 77 implementation of the Nelder-Mead algorithm. The source code itself is available in [26]. Many of the following implementations are based on this primary source code. We were not able to get the paper [27] itself.

On his website, John Burkardt gives a fortran 77 source code of the Nelder-Mead algorithm [28]. The following are the comments in the header of the source code.

```
c  Discussion:
c
c    This routine seeks the minimum value of a user-specified function.
c
c     Simplex function minimisation procedure due to Nelder+Mead(1965),
c     as implemented by O'Neill(1971, Appl.Statist. 20, 338-45), with
c     subsequent comments by Chambers+Ertel(1974, 23, 250-1), Benyon(1976,
```

```
c      25, 97) and Hill(1978, 27, 380-2)
c
c     The function to be minimized must be defined by a function of
c     the form
c
c       function fn ( x, f )
c       double precision fn
c       double precision x(*)
c
c     and the name of this subroutine must be declared EXTERNAL in the
c     calling routine and passed as the argument FN.
c
c     This routine does not include a termination test using the
c     fitting of a quadratic surface.
c
c   Modified:
c
c     27 February 2008
c
c   Author:
c
c     FORTRAN77 version by R ONeill
c     Modifications by John Burkardt
```

The "Bayesian Survival Analysis" book by Joseph G. Ibrahim, Ming-Hui Chen, and Debajyoti Sinha provides in [1] a fortran 77 implementation of the Nelder-Mead algorithm. The following is the header of the source code.

```
c      Simplex function minimisation procedure due to Nelder+Mead(1965),
c      as implemented by O'Neill(1971, Appl.Statist. 20, 338-45), with
c      subsequent comments by Chambers+Ertel(1974, 23, 250-1), Benyon(1976,
c      25, 97) and Hill(1978, 27, 380-2)
```

The O'Neill implementation uses a restart procedure which is based on a local axis by axis search for the optimality of the computed optimum.

B.7 Burkardt implementations

John Burkardt gives several implementations of the Nelder-Mead algorithm

- in fortran 77 [28]
- in Matlab by Jeff Borggaard [3].

B.8 NAG Fortran implementation

The NAG Fortran library provides the E04CCF/E04CCA routines [20] which implements the simplex optimization method. E04CCA is a version of E04CCF that has additional parameters in order to make it safe for use in multithreaded applications. As mentioned in the documentation, "The method tends to be slow, but it is robust and therefore very useful for functions that are subject to inaccuracies.". The termination criteria is based on the standard deviation of the function values of the simplex.

The specification of the cost function for E04CCA is:

```
SUBROUTINE FUNCT ( N, XC, FC, IUSER, RUSER)
```

where IUSER and RUSER and integer and double precision array, which allow the user to supply information to the cost function. An output routine, called MONIT is called once every iteration in E04CCF/E04CCA. It can be used to print out the current values of any selection of its parameters but must not be used to change the values of the parameters.

B.9 GSL implementation

The Gnu Scientific Library provides two Nelder-Mead implementations. The authors are Tuomo Keskitalo, Ivo Alxneit and Brian Gough. The size of the simplex is the root mean square sum of length of vectors from simplex center to corner points. The termination criteria is based on the size of the simplex.

The C implementation of the minimization algorithm is original. The communication is direct, in the sense that the specific optimization algorithm calls back the cost function. A specific optimization implementation provides four functions : "alloc", "free", "iterate" and "set". A generic optimizer is created by connecting it to a specific optimizer. The user must write the loop over the iterations, making successive calls to the generic "iterate" function, which, in turns, calls the specific "iterate" associated with the specific optimization algorithm.

The cost function can be provided as three function pointers

- the cost function f,
- the gradient g,
- both the cost function and the gradient.

Some additional parameters can be passed to these functions.

Bibliography

[1] optim1.f. http://www.stat.uconn.edu/~mhchen/survbook/example51/optim1.f.

[2] Johan Andersson and Linköpings Universitet. Multiobjective optimization in engineering design: Application to fluid power systems. Technical report, Department of Mechanical Engineering, Linköping University, 2001. https://polopoly.liu.se/content/1/c6/10/99/74/phdthesis.pdf, http://www.iei.liu.se/machine/johan-olvander.

[3] Jeff Borggaard. nelder_mead. January 2009. http://people.sc.fsu.edu/~burkardt/m_src/nelder_mead/nelder_mead.m.

[4] M. J. Box. A new method of constrained optimization and a comparison with other methods. *The Computer Journal*, pages 42–52, 1965.

[5] Yann Collette. Lolimot. http://sourceforge.net/projects/lolimot/.

[6] Yann Collette. Lolimot - optim_nelder_mead.sci. http://lolimot.cvs.sourceforge.net/viewvc/lolimot/scilab/optim/macros/optim_nelder_mead.sci?revision=1.1.1.1&view=markup.

[7] Ellen Fan. Global optimization of lennard-jones atomic clusters. Technical report, McMaster University, February 2002.

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

[9] J. A. Guin. Discussion and correspondence: modification of the complex method of constrained optimization. *The Computer Journal*, 10(4):416–417, February 1968.

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

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

[12] Lixing Han, Michael Neumann, and Jianhong Xu. On the roots of certain polynomials arising from the analysis of the nelder-mead simplex method. *Linear Algebra and its Applications*, 363:109–124, 2003.

[13] C. T. Kelley. *Iterative Methods for Optimization: Matlab Codes*. North Carolina State University. http://www4.ncsu.edu/~ctk/matlab_darts.html.

[14] 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.

[15] C. T. Kelley. *Iterative Methods for Optimization*, volume 19. SIAM Frontiers in Applied Mathematics, 1999. http://www.siam.org/books/textbooks/fr18_book.pdf.

[16] 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. http://link.aip.org/link/?SJE/9/112/1.

[17] The Mathworks. Matlab ? fminsearch. http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/access/helpdesk/help/techdoc/ref/fminsearch.html.

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

[19] 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.

[20] NAG. Nag fortran library routine document : E04ccf/e04cca. http://www.nag.co.uk/numeric/Fl/manual/xhtml/E04/e04ccf.xml.

[21] J. C. Nash. *Compact numerical methods for computers : linear algebra and function minimisation*. Hilger, Bristol, 1979.

[22] J.C. Nash. Gams - a19a20 - description. February 1980. http://gams.nist.gov/serve.cgi/Module/NASHLIB/A19A20/11238/.

[23] J.C. Nash. Gams - a19a20 - source code. February 1980. http://gams.nist.gov/serve.cgi/ModuleComponent/11238/Source/ITL/A19A20.

[24] Larry Nazareth and Paul Tseng. Gilding the lily: A variant of the nelder-mead algorithm based on golden-section search. *Comput. Optim. Appl.*, 22(1):133–144, 2002.

[25] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965. http://dx.doi.org/10.1093/comjnl/7.4.308.

[26] R. O'Neill. Algorithm as47 - fortran 77 source code. 1971. http://lib.stat.cmu.edu/apstat/47.

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

[28] R. O'Neill and John Burkardt. Gams - a19a20 - source code. 2008. http://people.sc.fsu.edu/~burkardt/f77_src/asa047/asa047.f.

[29] Parkinson and Hutchinson. An investigation into the efficiency of variants on the simplex method. *F. A. Lootsma, editor, Numerical Methods for Non-linear Optimization*, pages 115–135, 1972.

[30] Ronald N. Perry and Ronald N. Perry. A new method for numerical constrained optimization, 2001.

[31] D. Peters, H. Bolte, C. Marschner, O. Nüssen, and R. Laur. Enhanced optimization algorithms for the development of microsystems. *Analog Integr. Circuits Signal Process.*, 32(1):47–54, 2002.

[32] 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.

[33] W. H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, Second Edition*. 1992.

[34] Joel A. Richardson and J. L. Kuester. Algorithm 454: the complex method for constrained optimization. *Commun. ACM*, 16(8):487–489, 1973.

[35] 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.

[36] Kenneth D. Shere. Remark on algorithm 454 : The complex method for constrained optimization. *Commun. ACM*, 17(8):471, 1974.

[37] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461, 1962.

[38] M. B. Subrahmanyam. An extension of the simplex method to constrained nonlinear optimization. *J. Optim. Theory Appl.*, 62(2):311–319, 1989.

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