Nelder-Mead
Toolbox Manual
– Spendley et al. algorithm –

Contents

Notations

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

Fig. 1 : Notations used in this document

Chapter 1

Spendley's et al. method

In this chapter, we present Spendley, Hext and Himsworth algorithm [6] 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 in the behavior 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 behavior of the algorithm when the number of variables increases.

1.1 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.

1.1.1 Overview

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

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

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

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

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

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

The $\mathbf{v}_1$ vertex (resp. the $\mathbf{v}_{n+1}$ vertex) is called the *best* vertex (resp. *worst*), because it is associated with the lowest (resp. highest) function value. As we are going to see, the *next-to-worst* vertex $\mathbf{v}_n$ has a special role in this algorithm.

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

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

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

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

1.1.2 Algorithm

In this section, we analyze Spendley's et al algorithm, which is presented in figure 1.1.

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

$$\overline{\mathbf{x}}(n + 1) = \frac{1}{n} \sum_{i=1,n} \mathbf{v}_i. \tag{1.5}$$

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

$$\mathbf{x}_r = \mathbf{x}(\rho, n + 1) = (1 + \rho)\overline{\mathbf{x}}(n + 1) - \rho\mathbf{v}_{n+1} \tag{1.6}$$

We then compute the function value of the reflected point as $f_r = f(\mathbf{x}_r)$. If the function value f_r is better than the worst function value f_{n+1}, i.e. if $f_r < f_{n+1}$, then the worst vertex $\mathbf{v}_{n+1}$ is rejected from the simplex and the reflected point $\mathbf{x}_r$ is accepted. If the reflection point does not improve the function value f_{n+1}, we consider the centroid $\overline{\mathbf{x}}(n)$, i.e. the centroid where the next-to-worst vertex $\mathbf{v}_n$ has been excluded. We then consider the reflected point $\mathbf{x}'_r$, computed from the next-to-worst vertex $\mathbf{v}_n$ and the centroid $\overline{\mathbf{x}}(n)$. We compute the function value $f'_r = f(\mathbf{x}'_r)$. If the function value f'_r improves over the worst function value f_{n+1}, then the worst vertex $\mathbf{v}_{n+1}$ is rejected from the simplex and the new reflection point $\mathbf{x}'_r$ is accepted.

At that point of the algorithm, neither the reflection with respect to $\mathbf{v}_{n+1}$ nor the reflection with respect to $\mathbf{v}_n$ were able to improve over the worst function value f_{n+1}. Therefore, the algorithm shrinks the simplex toward the best vertex $\mathbf{v}_1$. That last step uses the shrink coefficient $0 < \sigma < 1$. The standard value for this coefficient is $\sigma = \frac{1}{2}$.

Compute an initial simplex S_0
Sorts the vertices S_0 with increasing function values
$S \leftarrow S_0$
while $\sigma(S) > tol$ **do**
   $\overline{x} \leftarrow \overline{\mathbf{x}}(n+1)$ {Compute the centroid}
   $\mathbf{x}_r \leftarrow \mathbf{x}(\rho, n+1)$ {Reflect with respect to worst}
   $f_r \leftarrow f(\mathbf{x}_r)$
   if $f_r < f_{n+1}$ **then**
     Accept $\mathbf{x}_r$
   else
     $\overline{x} \leftarrow \overline{\mathbf{x}}(n)$ {Compute the centroid}
     $\mathbf{x}'_r \leftarrow \mathbf{x}(\rho, n)$ {Reflect with respect to next-to-worst}
     $f'_r \leftarrow f(\mathbf{x}'_r)$
     if $f'_r < f_{n+1}$ **then**
       Accept $\mathbf{x}'_r$
     else
       Compute the vertices $\mathbf{v}_i = \mathbf{v}_1 + \sigma(\mathbf{v}_i - \mathbf{v}_1)$ for $i = 2, n+1$ {Shrink}
       Compute $f_i = f(\mathbf{v}_i)$ for $i = 2, n+1$
     end if
   end if
   Sort the vertices of S with increasing function values
end while

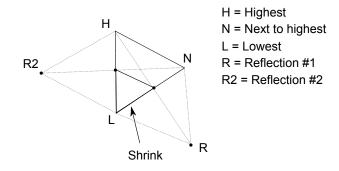**Fig. 1.1** : Spendley's et al. algorithm

H = Highest
N = Next to highest
L = Lowest
R = Reflection #1
R2 = Reflection #2

Fig. 1.2 : Spendley et al. simplex moves

H = Highest
N = Next to highest
L = Lowest
R = Reflection #1

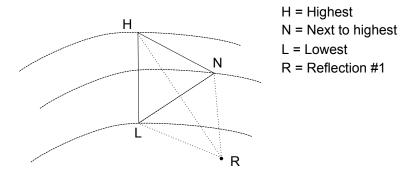**Fig. 1.3** : Spendley et al. simplex moves – Reflection with respect to highest point

1.1.3   Geometric analysis

The figure 1.2 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.

In Spendley's et al. original paper, the authors use a regular simplex, where the edges all have the same length. In practice, however, any non degenerate simplex can be used.

The various situations in which these moves are performed are presented in figures 1.3, 1.4 and 1.5.

The basic move is the reflection step, presented in figure 1.3 and 1.4. These two figures show that Spendley's 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 1.5.

In the situation of figure 1.5, 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.
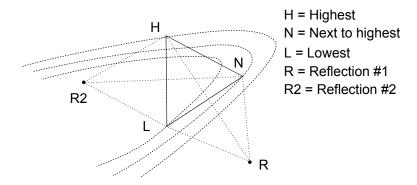
H = Highest
N = Next to highest
L = Lowest
R = Reflection #1
R2 = Reflection #2

Fig. 1.4 : Spendley et al. simplex moves – Reflection with respect to next-to-highest point. It may happen that the next iteration is a shrink step.
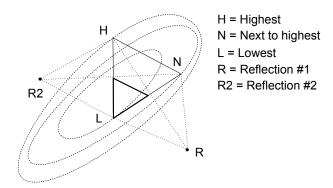
H = Highest
N = Next to highest
L = Lowest
R = Reflection #1
R2 = Reflection #2

Fig. 1.5 : Spendley et al. simplex moves – The shrink step is the only possible move.

1.1.4 General features of the algorithm

From the performance point of viewn when a reflection step is performed, only 1 or 2 function evaluations are required. Instead, when a shrink step is performed, there are n function evaluations required. In practice, reflection steps are performed when the simplex is away from the optimum. When the simplex is closer to the optimum, or enters in a narrow valley, shrink steps are used.

As stated in [5], the main feature of Spendley's et al. algorithm is that the simplex can vary in size, but not in shape. As we are going to see in the numerical experiments, this leads to a slow convergence when a narrow valley is encountered. In that situation, the shrink steps are required, which leads to a large number of iterations and function evaluations.

In fact, the Spendley's et al. algorithm is a pattern search algorithm [7]. This is a consequence of the fact that the search pattern used in the method is constant. Therefore, the design never degenerates. As stated in [7], "under very mild assumptions on f, these simple heuristics provide enough structure to guarantee global convergence. This is not the case for the Nelder-Mead algorithm, which might converge to non-stationnary points [4, 3, 1, 8]. In all cases, the difficulty is that a sequence of simplices produced by the Nelder-Mead simplex method can come arbitrarily close to degeneracy.

1.2 Numerical experiments

In this section, we present some numerical experiments with Spendley's et al. algorithm. The first numerical experiments involves one quadratic function in 2 dimensions. The second experiment is based on a badly scaled quadratic in 2 dimension. In the third experiment, we analyze the behavior of the algorithm with respect to the number of variables.

1.2.1 Quadratic function

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

$$f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2. \tag{1.7}$$

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

$$\sigma_+(S) < tol \times \sigma_+(S_0). \tag{1.8}$$

The oriented length $\sigma_+(S)$ is defined by

$$\sigma_+(S) = \max_{i=2,n+1} \|\mathbf{v}_i - \mathbf{v}_1\|_2 \tag{1.9}$$

where $\|.\|_2$ is the euclidian norm defined by

$$\|\mathbf{x}\|_2 = \sum_{i=1,n} x_i^2. \tag{1.10}$$

In this experiment, we use $tol = 10^{-8}$ as the relative tolerance on simplex size.

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 1.6.

Iterations	49
Function Evaluations	132
$\mathbf{x}_0$	$(2.0, 2.0)$
Relative tolerance on simplex size	10^{-8}
Exact $\mathbf{x}^\star$	$(0., 0.)$
Computed $\mathbf{x}^\star$	$(2.169e - 10, 2.169e - 10)$
Exact $f(\mathbf{x}^\star)$	$0.$
Computed $f(\mathbf{x}^\star)$	$4.706e - 20$

Fig. 1.6 : 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 1.7. The method use reflections in the early iterations. Then there is no possible improvement using reflections and shrinking is necessary. That behavior is an illustration of the discretization which has already been discussed.

The figure 1.8 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 1.9.

1.2.2 Badly scaled quadratic function

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

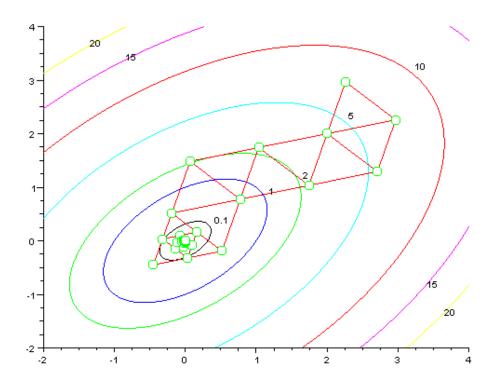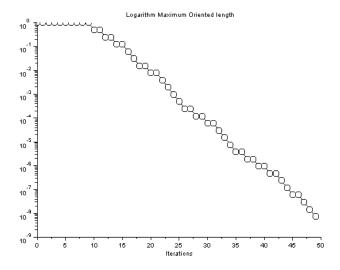$$f(x_1, x_2) = ax_1^2 + x_2^2, \tag{1.11}$$

Fig. 1.7 : Spendley et al. numerical experiment – History of simplex

Fig. 1.8 : Spendley et al. numerical experiment – History of logarithm of the size of the simplex
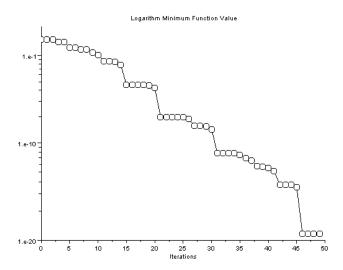
Fig. 1.9 : Spendley et al. numerical experiment – History of logarithm of function

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. Indeed, let us compute the Hessian matrix associated with the cost function. We have

$$\mathbf{H} = \begin{pmatrix} 2a & 0 \\ 0 & 2 \end{pmatrix}. \tag{1.12}$$

Therefore, the eigenvalues of the Hessian matrix are $2a$ and 2, which are stricly positive if $a > 0$. Hence, the cost function is stricly convex and there is only one global solution, that is $\mathbf{x}^\star = (0, 0, \dots, 0)^T$. The ratio between these two eigenvalues is a. This leads to an elongated valley, which is extremely narrow when a is large.

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

$$\sigma_+(S) < tol \times \sigma_+(S_0). \tag{1.13}$$

In this experiment, we use $tol = 10^{-8}$ as the relative tolerance on simplex size.

We set the maximum number of function evaluations to 400. The initial simplex is a regular simplex with length unity.

The following Scilab script allows to perform the optimization.

```
a = 100;
function y = quadratic (x)
  y = a * 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 1.6, where the experiment is presented for $a = 100$. We can check that the number of function evaluations is equal to its maximum limit, even if the value of the function at optimum is very inaccurate ($f(\mathbf{x}^\star) \approx 0.08$).

Iterations	340
Function Evaluations	400
a	100.0
$\mathbf{x}_0$	$(10.0, 10.0)$
Relative tolerance on simplex size	10^{-8}
Exact $\mathbf{x}^\star$	$(0., 0.)$
Computed $\mathbf{x}^\star$	$(0.001, 0.2)$
Computed $f(\mathbf{x}^\star)$	0.08

Fig. 1.10 : 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 1.11. The method use reflections in the early iterations. Then there is no possible improvement using reflections, so that shrinking is necessary. But the repeated shrink steps makes the simplex very small, leading to a large number of iterations. This is a limitation of the method, which is based on a simplex which can vary its size, but not its shape.

In figure 1.12, we analyze the behavior 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 $a > 10$

1.2.3 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, as presented by Han & Neumann in [2]. We emphasize, though, that Han & Neumann present their numerical experiment with the Nelder-Mead algorithm, while we present in this section the Spendley et al. algorithm. The function we try to minimize is the following quadratic function in n-dimensions

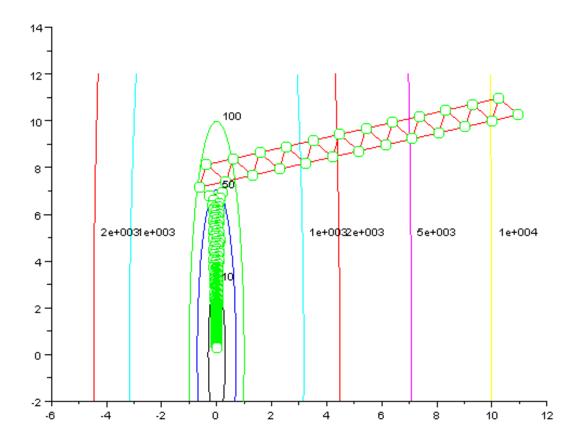$$f(\mathbf{x}) = \sum_{i=1,n} x_i^2. \tag{1.14}$$

Fig. 1.11 : 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 $f(\mathbf{x}^\star)$
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. 1.12 : Numerical experiment with Spendley's et al. method on a badly scaled quadratic function

The initial guess is the origin $\mathbf{x}_0 = (0, 0, \ldots, 0)^T$, which is also the global solution of the problem. We have $f(\mathbf{x}_0) = 0$ 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. An absolute termination criteria on the size of the simplex is used, that is, the algorithm is stopped when the inequality $\sigma_+(S_k) \leq 10^{-8}$ is satisfied.

For this test, we compute the rate of convergence as presented in Han & Neuman [2]. This rate is defined as

$$\rho(S_0, n) = \lim \sup_{k \to \infty} \left(\prod_{i=0, k-1} \frac{\sigma(S_{i+1})}{\sigma(S_i)} \right)^{1/k}, \tag{1.15}$$

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. This definition implies

$$\rho(S_0, n) = \lim \sup_{k \to \infty} \left(\frac{\sigma(S_k)}{\sigma(S_0)} \right)^{1/k}, \tag{1.16}$$

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

$$\rho(S_0, n, k) = \left(\frac{\sigma(S_k)}{\sigma(S_0)} \right)^{1/k}. \tag{1.17}$$

The following Scilab script allows to perform the optimization.

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

The figure 1.14 presents the number of function evaluations depending on the number of variables. We can see that the number of function evaluations increases approximately linearly with the dimension of the problem in figure 1.15. A rough rule of thumb is that, for $n = 1, 20$, the number of function evaluations is equal to $30n$: most iterations are shrink steps and approximately 30 iterations are required, almost independently of n.

The table 1.14 also shows the interesting fact that the convergence rate is almost constant and very close to $1/2$. This is a consequence of the shrink steps, which are dividing the size of the simplex at each iteration by 2.

1.3   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	#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. 1.13 : Numerical experiment with Spendley et al method on a generalized quadratic function – Number of iterations and types of steps performed

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

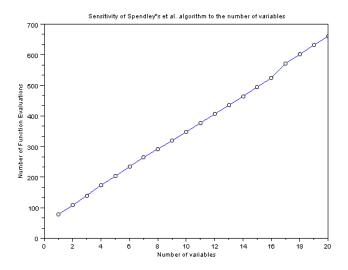**Fig. 1.14** : Numerical experiment with Spendley et al. method on a generalized quadratic function

Fig. 1.15 : 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. In this experiment, the rate of convergence is close to $1/2$ and the number of function evaluations is a linear function of the number of variables (approximately $30n$).

Bibliography

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

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

[3] 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.

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

[5] A. Singer and J. Nelder. Nelder-mead algorithm. *Scholarpedia*, 4(7):2928, 2009.

[6] 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.

[7] Virginia Torczon and Michael W. Trosset. From evolutionary operation to parallel direct search: Pattern search algorithms for numerical optimization. *Computing Science and Statistics*, 29:396–401, 1998.

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

Index