Nelder-Mead Toolbox Manual – Introduction –

Version 0.2
September 2009

Michaël BAUDIN

Contents

Chapter 1

Introduction

In this introductory chapter, we make an overview of simplex-based algorithms. We present the main features of the *neldermead* component, and show how to use the component with a simple example.

1.1 Overview

The Nelder-Mead simplex algorithm [2], 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 [5], 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 [4] (1962), which uses a fixed shape simplex. Historically, the algorithm created by Nelder and Mead was designed as an improvement on Spendley's et al. algorithm. The Box complex algorithm [1] (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.

- The "neldermead" component provides various simplex-based algorithms and manages for Nelder-Mead specific settings, such as the method to compute the initial simplex and the specific termination criteria.
- The "fminsearch" component 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.
- The "optimset" and "optimget" components provide Scilab commands to emulate their Matlab counterparts.
- The "nmplot" component provides features to produce directly output pictures for Nelder-Mead algorithm.

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

- The "optimbase" component 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 logging system, various termination criteria, the cost function, etc...

- The "optimsimplex" component 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...) and many algorithms to perform reflections and shrinkages.

The following is a list of features the Nelder-Mead 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
  - 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 :
  - the history of function values,
  - the history of optimum point,
  - the history of simplices,
  - the history of termination criterias,
- provide a plot command which allows to graphically see the history of the simplices toward the optimum,
- provide query functions for
  - the status of the optimization process,

  - the number of iterations,
  - the number of function evaluations,
  - the status of execution,
  - the function value at initial point,
  - the 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.

1.2 How to use the Toolbox

The design of the toolbox is based on the creation of a new token by the *neldermead_new* function. The Nelder-Mead object associated with this token can then be configured with *neldermead_configure* and queried with *neldermead_cget*. For example, 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...

Once the optimization is finished, the *neldermead_destroy* function deletes the object.

1.3 An example

In the following example, we search the minimum of the 2D Rosenbrock function [3], defined by

$$f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2 \tag{1.6}$$

The following Scilab script allows to find the solution of the problem. We begin by defining the function *rosenbrock* which computes the Rosenbrock function. The traditionnal initial guess $(-1.2, 1.0)$ is used, which corresponds to the "-x0" key. The initial simplex is computed along the axes with a length equal to 0.1. We want to use the Nelder-Mead algorithm with variable simplex size is used, which corresponds to the "variable" value of the "-method" option. 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,"-numberofvariables",2);
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")
status = neldermead_get(nm,"-status")
nm = neldermead_destroy(nm);
```

This produces the following output.

```
-->nm = neldermead_search(nm);
Function Evaluation #1 is [24.2] at [-1.2 1]
Function Evaluation #1 is [24.2] at [-1.2 1]
Function Evaluation #2 is [8.82] at [-1.1 1]
Function Evaluation #3 is [16.4] at [-1.2 1.1]
Step #1 : order
=============================================================
Iteration #1 (total = 1)
Function Eval #3
Xopt : -1.1 1
Fopt : 8.820000e+000
DeltaFv : 1.538000e+001
Center : -1.1666667 1.0333333
Size : 1.414214e-001
Vertex #1/3 : fv=8.820000e+000, x=-1.100000e+000 1.000000e+000
Vertex #2/3 : fv=1.640000e+001, x=-1.200000e+000 1.100000e+000
Vertex #3/3 : fv=2.420000e+001, x=-1.200000e+000 1.000000e+000
Reflect
xbar=-1.15 1.05
Function Evaluation #4 is [5.62] at [-1.1 1.1]
xr=[-1.1 1.1], f(xr)=5.620000
Expand
Function Evaluation #5 is [4.428125] at [-1.05 1.15]
xe=-1.05 1.15, f(xe)=4.428125
  > Perform Expansion
Sort
[...]
=============================================================
Iteration #56 (total = 56)
```

```
Function Eval #98
Xopt : 0.6537880 0.4402918
Fopt : 1.363828e-001
DeltaFv : 1.309875e-002
Center : 0.6788120 0.4503999
Size : 6.945988e-002
Vertex #1/3 : fv=1.363828e-001, x=6.537880e-001 4.402918e-001
Vertex #2/3 : fv=1.474625e-001, x=7.107987e-001 4.799712e-001
Vertex #3/3 : fv=1.494816e-001, x=6.718493e-001 4.309367e-001
Reflect
xbar=0.6822933 0.4601315
Function Evaluation #99 is [0.1033237] at [0.6927374 0.4893262]
xr=[0.6927374 0.4893262], f(xr)=0.103324
Expand
Function Evaluation #100 is [0.1459740] at [0.7031815 0.5185210]
xe=0.7031815 0.5185210, f(xe)=0.145974
  > Perform reflection
Sort
================================================================
Iteration #57 (total = 57)
Function Eval #100
Xopt : 0.6927374 0.4893262
Fopt : 1.033237e-001
DeltaFv : 4.413878e-002
Center : 0.6857747 0.4698631
Size : 6.262139e-002
Vertex #1/3 : fv=1.033237e-001, x=6.927374e-001 4.893262e-001
Vertex #2/3 : fv=1.363828e-001, x=6.537880e-001 4.402918e-001
Vertex #3/3 : fv=1.474625e-001, x=7.107987e-001 4.799712e-001
Terminate with status : maxfuneval
-->xopt = neldermead_get(nm,"-xopt")
 xopt  =

    0.6927374
    0.4893262

-->fopt = neldermead_get(nm,"-fopt")
 fopt  =

    0.1033237

-->status = neldermead_get(nm,"-status")
 status  =

 maxfuneval
```

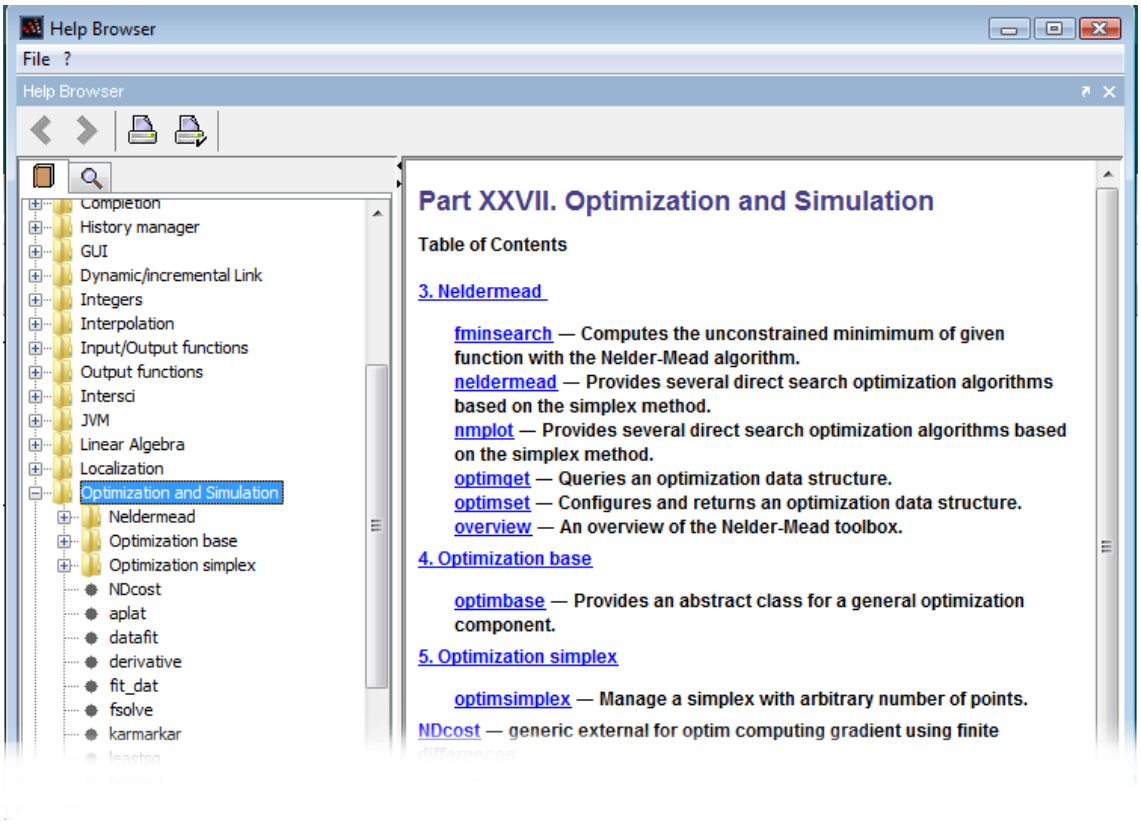

Fig. 1.1 : Built-in help for the Nelder-Mead component

1.4 Help, demonstrations and unit tests

For a complete presentation of the functions and options, the reader should consult the help which is provided with the component. The main menu of the help associated with the optimization module is presented in figures 1.1 and 1.2. The corresponding pages provide a complete documentation for the corresponding functions, as well as many sample uses.

Several demonstrations are provided with the component. These are available from the "Demonstration" menu of the Scilab console and are presented in figure 1.3.

The following script shows where the demonstration scripts are available from the Scilab installation directory.

```
1 -->cd SCI/modules/optimization/demos/neldermead
2  ans  =
3
4  D:\Programs\SCFD8E~1\modules\optimization\demos\neldermead
5
6 -->ls *.sce
7  ans  =
8
9 !nmplot_rosenbrock.sce          !
```

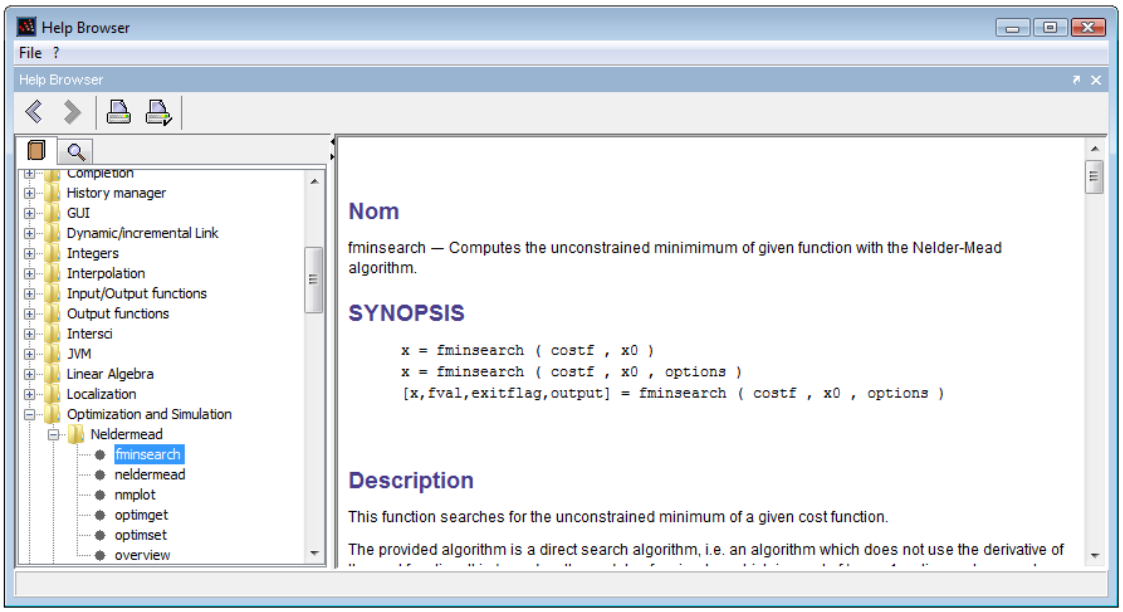

Fig. 1.2 : Built-in help for the *fminsearch* function

```
10 !                                !
11 !nmplot_rosenbrock.fixed.sce     !
12 !                                !
13 !nmplot_quadratic.fixed.sce      !
14 !                                !
15 !nmplot_mckinnon2.sce            !
16 !                                !
17 !nmplot_mckinnon.sce             !
18 !                                !
19 !nmplot_han2.sce                 !
20 !                                !
21 !nmplot_han1.sce                 !
22 !                                !
23 !nmplot_boxproblemA.sce          !
24 !                                !
25 !neldermead_rosenbrock.sce       !
26 !                                !
27 !neldermead.dem.sce              !
28 !                                !
29 !fminsearch.sce                  !
```

These components were developped based on unit tests, which are provided with Scilab. These unit tests are located in the "SCI/modules/optimization/tests/unit_tests" directory, under the "neldermead", "optimsimplex" and "optimbase" directories. Each unit test correspond to a .tst file. These tests are covering most (if not all) the features provided by the components. This is why there are a good source of information on how to use the functions.

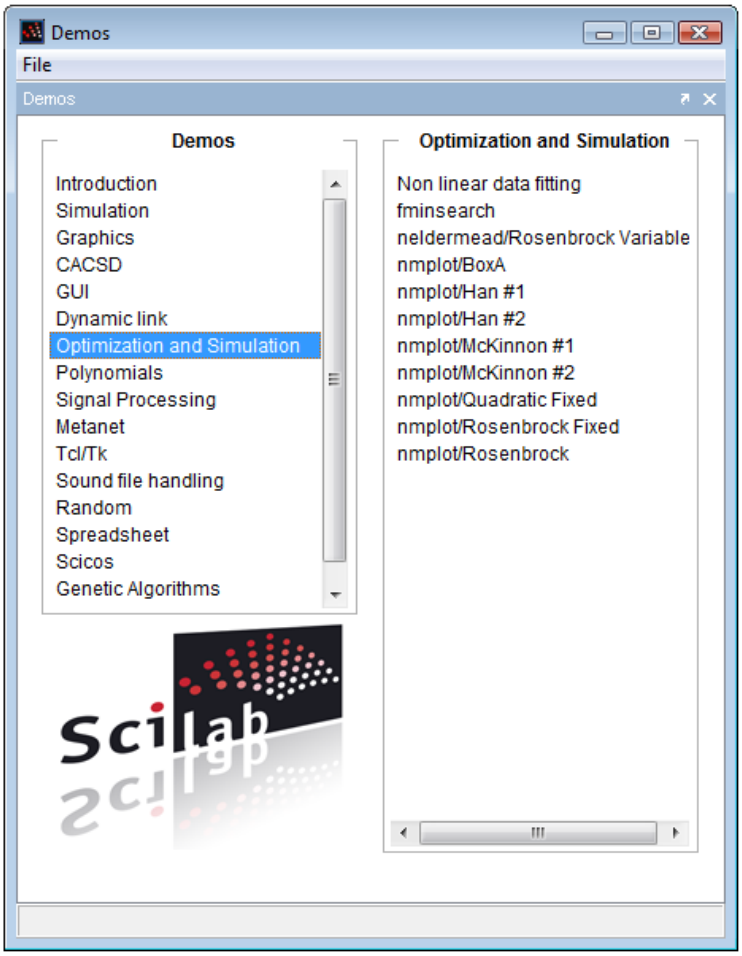

Fig. 1.3 : Built-in demonstration scripts for the Nelder-Mead component

Bibliography

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

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

[3] 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.

[4] 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.

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