The neldermead Package

Sébastien Bihorel

September 16, 2010

neldermead is a R port of a module originally developped for Scilab version 5.2.1 by Michael Baudin (INRIA - DIGITEO). Information about this software can be found at `www.scilab.org`. The following documentation as well as the content of the functions .Rd files are adaptations of the documentation provided with the original Scilab neldermead module.

neldermead currently does not include any adaptation of the Scilab 'nmplot' function series that is available in the original neldermead module.

1 Overview

1.1 Description

The goal of this toolbox is to provide several direct search optimization algorithms based on the simplex method. The optimization problem to solve is the minimization of a cost function, with bounds and nonlinear constraints.

$$
\begin{array}{ll}
minf(x) & \\
l_i \leq x_i \leq h_i, & i = 1, n \\
g_j(x) \geq 0, & j = 0, nbineq
\end{array}
$$

where f is the cost function, x is the vector of parameter estimates, l and h are vectors of lower and upper bounds for the parameter estimates, n is the number of parameters and $nbineq$ the number of inequality constraints $g(x)$.

The provided algorithms are direct search algorithms, i.e. algorithms which do not use the derivative of the cost function. They are based on the update of a simplex, which is a set of $k \geq n + 1$ vertices, where each vertex is associated with one point and one function value.

The following algorithms are available:

- The fixed shape simplex method of Spendley, Hext and Himsworth: this algorithm solves an unconstrained optimization problem with a fixed shape simplex made of $k = n + 1$ vertices.
- The variable shape simplex method of Nelder and Mead: this algorithm solves an unconstrained optimization problem with a variable shape simplex made of $k = n + 1$ vertices [3].
- Box's complex method: this algorithm solves an constrained optimization problem with a variable shape simplex made of an arbitrary k number of vertices ($k = 2n$ is recommended by Box).

1.2 Basic object

The basic object used by the **neldermead** package to store the configuration settings and the history of an optimization is a 'neldermead' object, i.e. a list typically created by `neldermead.new` and having a strictly defined structure (see `?neldermead.new` for more details).

1.3 The cost function

The `function` element of the neldermead object allows to configure the cost function. The cost function is used to compute the objective function value `f`. If the `nbineqconst` element of the neldermead object is configured to a non-zero value, the cost function must also compute the value of the nonlinear, positive, inequality constraints `c`. The cost function can also take as input/output an additional argument, if the `costfargument` element is configured. The function should be defined as described in `vignette('optimbase',package='optimbase')`:

```
costf <- function(x, index, fmsfundata)
```

where

`x`: is the current point, as a column vector,

`index`: (optional), an integer representing the value to compute, and

`fmsfundata`: an user-provided input/output argument.

The index input parameter tells the cost function what to return as output arguments (as described in `vignette('optimbase',package='optimbase')`. It has the following meaning:

index = 2: compute `f`, the value of the cost function

index = 5: compute `c`, the value of the non-linear, positive, inequality constraints

index = 6: compute `f` and `c`

The `fmsdata` argument is both input and output. This feature may be used in the situation where the cost function has to update its environment from call to call. Its simplest use is to count the number of calls to the cost function, but this feature is already available directly. Consider the more practical situation where the optimization requires the execution of an underlying Newton method (a chemical solver for example). This Newton method requires an initial guess x0. If the initial guess for this underlying Newton method is kept constant, the Newton method may have problems to converge when the current optimization point get far away from the its initial point. If a `costfargument` element is defined in the neldermead object, it can be passed to the cost function as the `fmsdata` argument. In this case, the initial guess for the Newton method can be updated so that it gets the value of the previous call. This way, the Newton method will have less problems to converge and the cost function evaluation may be faster.

We now present how the feature works. Everytime the cost function is called back, the `costfargument` element is passed to the cost function as an input argument. If the cost function modifies its content in the output argument, the content of the `costfargument` element is updated accordingly. Once the optimization is performed, the user may call the `neldermead.cget` function and get back an updated `costfargument` content.

1.4 The output function

The `outputcommand` element of the neldermead object allows to configure a command which is called back at the start of the optimization, at each iteration and at the end of the optimization. The output function must be defined as follows:

```
outputcmd <- function(state, data, myobj)
```

where

`state`: is a string representing the current state of the algorithm. Available values are 'init', 'iter', and 'done',

`data`: a list containing at least the following entries:

- `x`: the current optimum,
- `fval`: the current function value,
- `iteration`: the current iteration index,
- `funccount`: the number of function evaluations,
- `simplex`: the current simplex,
- `step`: the previous step in the algorithm. The following values are available: 'init', 'done', 'reflection', 'expansion', 'insidecontraction', 'outsidecontraction', 'reflectionnext', and 'shrink',

`myobj`: a user-defined parameter. This input parameter is defined with the `outputcommandarg` element of the neldermead object.

The output function may be used when debugging the specialized optimization algorithm, so that a verbose logging is produced. It may also be used to write one or several report files in a specialized format (ASCII, LaTeX, Excel, etc...). The user-defined parameter may be used in that case to store file names or logging options.

The `data` list argument may contain more fields than the current presented ones. These additionnal fields may contain values which are specific to the specialized algorithm, such as the simplex in a Nelder-Mead method, the gradient of the cost function in a BFGS method, etc...

1.5 Termination

The current package takes into account several generic termination criteria. The following termination criteria are enabled by default:

- maxiter,
- maxfunevals,
- tolxmethod,
- tolsimplexizemethod.

The `neldermead.termination` function uses a set of rules to compute if the termination occurs and sets optimization status to one of the following: 'continue', 'maxiter', 'maxfunevals', 'tolf', 'tolx', 'tolsize', 'tolsizedeltafv', 'kelleystagnation', 'tolboxf' or 'tolvariance'. The value of the status may also be a user-defined string, in the case where a user-defined termination function has been set.

The following set of rules is examined in this order.

- By default, the status is 'continue' and the terminate flag is FALSE.
- The number of iterations is examined and compared to the `maxiter` element of the neldermead object: if `iterations` $\geq$ `maxiter`, then the status is set to 'maxiter' and terminate is set to TRUE.
- The number of function evaluations is examined and compared to the `maxfunevals` elements: if `funevals` $\geq$ `maxfunevals`, then the status is set to 'maxfuneval' and terminate is set to TRUE.
- The tolerance on function value is examined depending on the value of the `tolfunmethod`.

  FALSE: then the criteria is just ignored,

  TRUE: if `|currentfopt| < tolfunrelative` $\cdot$ `|previousfopt| + tolfunabsolute`, then the status is set to 'tolf' and terminate is set to TRUE.

  The relative termination criteria on the function value works well if the function value at optimum is near zero. In that case, the function value at initial guess fx0 may be used as previousfopt. This criteria is sensitive to the `tolfunrelative` and `tolfunabsolute` elements. The absolute termination criteria on the function value works if the user has an accurate idea of the optimum function value.
- The tolerance on x is examined depending on the value of the `tolxmethod` element.

  FALSE: then the criteria is just ignored,

  TRUE: if `norm(currentxopt - previousxopt) < tolxrelative` $\cdot$ `norm(currentxopt) + tolxabsolute`, then the status is set to 'tolx' and terminate is set to TRUE.

  This criteria is sensitive to the `tolxrelative` and `tolxabsolute` elements. The relative termination criteria on x works well if x at optimum is different from zero. In that case, the condition measures the distance between two iterates. The absolute termination criteria on x works if the user has an accurate idea of the scale of the optimum x. If the optimum x is near 0, the relative tolerance will not work and the absolute tolerance is more appropriate.
- The tolerance on simplex size is examined depending on the value of the `tolsimplexizemethod` element.

  FALSE: then the criteria is just ignored,

  TRUE: if `ssize < tolsimplexizerelative` $\cdot$ `simplexsize0 + tolsimplexizeabsolute`, where `simplexsize0` is the size of the simplex at iteration 0, then the status is set to 'tolsize' and terminate is set to TRUE.

  The size of the simplex is computed from the 'sigmaplus' method of the **optimsimplex** package. This criteria is sensitive to the `tolsimplexizeabsolute` and the `tolsimplexizerelative` elements.
- The absolute tolerance on simplex size and absolute difference of function value is examined depending on the value of the `tolssizedeltafvmethod` element.

  FALSE: then the criteria is just ignored,

TRUE: if both the following conditions `ssize < tolsimplexizeabsolute` and `shiftfv < toldeltafv` are true where `ssize` is the current simplex size and `shiftfv` is the absolute value of the difference of function value between the highest and lowest vertices, then the status is set to 'tolsizedeltafv' and terminate is set to TRUE.

- The stagnation condition based on Kelley sufficient decrease condition is examined depending on the value of the `kelleystagnationflag` element.

  FALSE: then the criteria is just ignored,

  TRUE: if `newfvmean` $\leq$ `oldfvmean - alpha` $\cdot$ `t(sg)` $\cdot$ `sg` where `newfvmean` (resp. `oldfvmean`) is the function value average in the current iteration (resp. in the previous iteration), then the status is set to 'kelleystagnation' and terminate is set to TRUE. Here, `alpha` is a non-dimensional coefficient and `sg` is the simplex gradient.

- The termination condition suggested by Box is examined depending on the value of the `box-termination` element.

  FALSE: then the criteria is just ignored,

  TRUE: if both the following conditions `shiftfv < boxtolf` and `boxkount == boxnbmatch` are true, where `shiftfv` is the difference of function value between the best and worst vertices, and `boxkount` is the number of consecutive iterations where this criteria is met, then the status is set to 'tolboxf' and terminate is set to TRUE. Here, the `boxtolf` parameter is the value associated with the `boxtolf` element of the neldermead object and is a user-defined absolute tolerance on the function value. The `boxnbmatch` parameter is the value associated with the `boxnbmatch` element and is the user-defined number of consecutive match.

- The termination condition based on the variance of the function values in the simplex is examined depending on the value of the `tolvarianceflag` element.

  FALSE: then the criteria is just ignored,

  TRUE: if `var < tolrelativevariance` $\cdot$ `variancesimplex0 + tolabsolutevariance`, where `var` is the variance of the function values in the simplex, then the status is set to 'tolvariance' and terminate is set to TRUE. Here, the `tolrelativevariance` parameter is the value associated with the `tolrelativevariance` element of the neldermead object and is a user-defined relative tolerance on the variance of the function values. The `tolabsolutevariance` parameter is the value associated with the `tolabsolutevariance` element and is the user-defined absolute tolerance of the variance of the function values.

- The user-defined termination condition is examined depending on the value of the `myterminateflag` element.

  FALSE: then the criteria is just ignored,

  TRUE: if the `term` boolean output argument returned by the termination function is TRUE, then the status is set to the user-defined status and terminate is set to TRUE.

1.6 Kelley's stagnation detection

The stagnation detection criteria suggested by Kelley is based on a sufficient decrease condition, which requires a parameter alpha > 0 to be defined [1]. The `kelleynormalizationflag` element of

the neldermead object allows to configure the method to use to compute this alpha parameter. Two methods are available, where each method corresponds to a different paper by Kelley:

constant: in 'Detection and Remediation of Stagnation in the Nelder-Mead Algorithm Using a Sufficient Decrease Condition', Kelley uses a constant alpha, with the suggested value 1.e-4, which is the typical choice for line search method.

normalized: in 'Iterative Methods for Optimization', Kelley uses a normalized alpha, computed from the following formula: `alpha = alpha0 · sigma0 / nsg`, where `sigma0` is the size of the initial simplex and `nsg` is the norm of the simplex gradient for the initial guess point.

1.7 O'Neill's factorial optimality test

In 'Algorithm AS47 - Function minimization using a simplex procedure', O'Neill presents a fortran 77 implementation of the simplex method [6]. A factorial test is used to check if the computed optimum point is a local minimum. If the `restartdetection` element of the neldermead object is set to 'oneill', that factorial test is used to see if a restart should be performed.

1.8 Implementation notes of the method of Spendley *et al.*

The original paper may be implemented with several variations, which might lead to different results [7]. This section defines what algorithmic choices have been used in the present package.

The paper states the following rules.

- 'Rule 1. Ascertain the lowest reading y, of yi ... yk+1 Complete a new simplex Sp by excluding the point Vp corresponding to y, and replacing it by V* defined as above.'
- 'Rule 2. If a result has occurred in (k + 1) successive simplexes, and is not then eliminated by application of Rule 1, do not move in the direction indicated by Rule 1, or at all, but discard the result and replace it by a new observation at the same point.'
- 'Rule 3. If y is the lowest reading in So , and if the next observation made, y* , is the lowest reading in the new simplex S , do not apply Rule 1 and return to So from Sp . Move out of S, by rejecting the second lowest reading (which is also the second lowest reading in So).'

We implement the following 'rules' of the Spendley *et al.* method:

- Rule 1 is strictly applied, but the reflection is done by reflection of the high point, since we minimize a function instead of maximizing it, like Spendley.
- Rule 2 is NOT implemented, as we expect that the function evaluation is not subject to errors.
- Rule 3 is applied, i.e. reflection with respect to next to the high point. The original paper does not mention any shrink step. When the original algorithm cannot improve the function value with reflection steps, the basic algorithm stops. In order to make the current implementation of practical value, a shrink step is included, with shrinkage factor sigma. This perfectly fits into to the spirit of the original paper. Notice that the shrink step makes the rule #3 (reflection with respect to next-to-worst vertex) unnecessary. Indeed, the minimum required steps are the reflection and shrinkage. Nevertheless, the rule #3 has been kept in order to make the algorithm as close as it can be to the original.

1.9 Implementation notes on the method of Nelder and Mead

The purpose of this section is to analyse the current implementation of Nelder-Mead's algorithm. The algorithm that we use is described in 'Iterative Methods for Optimization' by Kelley.

The original paper uses a 'greedy' expansion, in which the expansion point is accepted whatever its function value. The current implementation, as most implementations, uses the expansion point only if it improves over the reflection point, that is,

- if fe<fr, then the expansion point is accepted,
- if not, the reflection point is accepted.

The termination criteria suggested by Nelder and Mead is based on an absolute tolerance on the standard deviation of the function values in the simplex. We provide this original termination criteria with the `tolvarianceflag` element of the neldermead object, which is disabled by default.

1.10 Box's complex algorithm implementation notes

In this section, we analyse the current implementation of Box's complex method [5]. The initial simplex can be computed as in Box's paper, but this may not be safe. In his paper, Box suggests that if a vertex of the initial simplex does not satisfy the non linear constraints, then it should be 'moved halfway toward the centroid of those points already selected'. This behaviour is available when the `scalingsimplex0` element of the neldermead object is set to 'tocenter'. It may happen, as suggested by Guin [2], that the centroid is not feasible if the constraints are not convex. In this case, the initial simplex cannot be computed. This is why we provide the 'tox0' option, which allows to compute the initial simplex by scaling toward the initial guess, which is always feasible.

In Box's paper, the scaling into the non linear constraints is performed 'toward' the centroid, that is, by using a scaling factor equal to 0.5. This default scaling factor might be sub-optimal in certain situations. This is why we provide the `boxineqscaling` element, which allows to configure the scaling factor.

In Box's paper, whether we are concerned with the initial simplex or with the simplex at a given iteration, the scaling for the non linear constraints is performed without end. This is because Box's hypothesis is that 'ultimately, a satisfactory point will be found'. As suggested by Guin, if the process fails, the algorithm goes into an infinite loop. In order to avoid this, we perform the scaling until a minimum scaling value is reached, as defined by the `guinalphamin` element.

We have taken into account the comments by Guin, but it should be emphasized that the current implementation is still as close as possible to Box's algorithm and is not Guin's algorithm. More precisely, during the iterations, the scaling for the non linear constraints is still performed toward the centroid, be it feasible or not.

1.11 User-defined algorithm

The `mymethod` element of the neldemead object allows to configure a user-defined simplex-based algorithm. The reason for this option is that many simplex-based variants of Nelder-Mead's algorithm have been developped over the years, with specific goals. While it is not possible to provide them all, it is very convenient to use the current structure without being forced to make many developments.

The value of the `mymethod` element is expected to be a R function with the following structure:

```
> myalgorithm <- function(this) {
+     ...
```

```
+     return(this)
+ }
```

where `this` is the current neldermead object.

In order to use the user-defined algorithm, the `method` element must be set to 'mine'. In this case, the component performs the optimization exactly as if the user-defined algorithm was provided by the component.

The user interested in that feature may use the internal scripts provided in the distribution as templates and tune his own algorithm from that point. There is of course no warranty that the user-defined algorithm improves on the standard algorithm, so that users use this feature at their own risks.

1.12 User-defined termination

Many termination criteria are found in the literature. Users who aim at reproducing the results exhibited in a particular paper may find that that none of the provided termination criteria match the one which is used in the paper. It may also happen that the provided termination criteria are not suitable for the specific test case. In those situation the `myterminate` element of the neldermead object allows to configure a user-defined termination function. The value of the `myterminate` element is expected to be a R function with the following structure:

```
> mystoppingrule <- function(this, simplex) {
+     ...
+     return(list(this = this, terminate = terminate, status = status))
+ }
```

where `this` is the current neldermead object and `simplex` is the current simplex. The `terminate` output argument is a logical flag which is FALSE if the algorithm must continue and TRUE if the algorithm must stop. The `status` output argument is a string which is associated with the current termination criteria.

In order to enable the use of the user-defined termination function, the value of the `myterminateflag` element must be set to TRUE in the neldermead object. At each iteration, if the `myterminateflag` element has been set to TRUE, the user-defined termination is called. If the terminate output argument is TRUE, then the algorithm is stopped. In that case, the value of the `status` element of the `neldermead.get` function output is the value of the `status` output argument of the user-defined termination function.

2 Examples

We present in this section basic examples illustrating the use of **neldermead** functions to optimize unconstrained or constrained systems. More complex examples are described in a Scilab-based document written by Michael Baudin and available at `http://forge.scilab.org/index.php/p/docneldermead/`. Because the R port of the Scilab **neldermead** module is almost literal, the user should be able to reproduce the described examples in R with minimal adaptations.

2.1 Example 1: basic use

In the following example, we solve a simple quadratic test case. We begin by defining the cost function, which takes 3 input arguments and returns the value of the objective function as the

f element of a list. The standard starting point [-1.2 1.0] is used. neldermead.new creates a new neldermead object. Then we use neldermead.configure to configure the parameters of the problem. We use all default settings and perform the search for the optimum. neldermead.get is finally used to retrieve the optimum parameters.

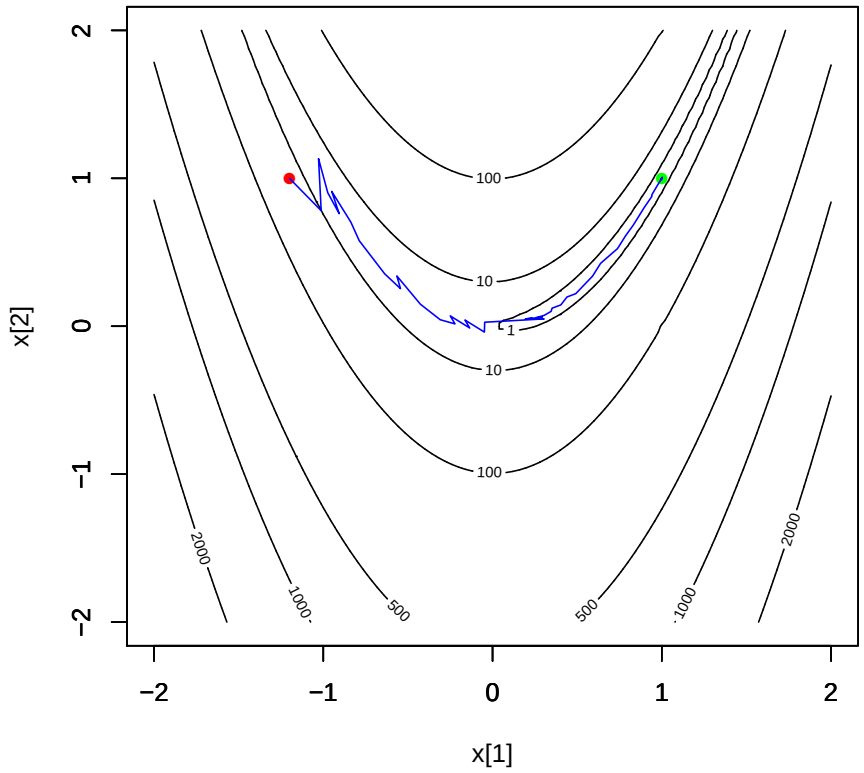
```
> quadratic <- function(x = NULL, index = NULL, fmsfundata = NULL) {
+     return(list(f = x[1]^2 + x[2]^2, g = c(), c = c(), gc = c(),
+         index = index, this = list(costfargument = fmsfundata)))
+ }
> x0 <- transpose(c(1, 1))
> nm <- neldermead.new()
> nm <- neldermead.configure(nm, "-numberofvariables", 2)
> nm <- neldermead.configure(nm, "-function", quadratic)
> nm <- neldermead.configure(nm, "-x0", x0)
> nm <- neldermead.search(nm)
> transpose(neldermead.get(nm, "-xopt"))

              [,1]          [,2]
[1,] -1.010582e-08 -1.768891e-07
```

2.2 Example 2: customized use

In the following example, we solve the Rosenbrock test case. We begin by defining the Rosenbrock function, which takes 3 input arguments and returns the value of the objective function. The standard starting point [-1.2 1.0] is used. neldermead.new creates a new neldermead object. Then we use neldermead.configure to configure the parameters of the problem. The initial simplex is computed from the axes and the single length 1.0 (this is the default, but is explicitely written here as an example). The variable simplex algorithm by Nelder and Mead is used, which corresponds to the -method 'variable' option. neldermead.search performs the search for the minimum. Once the minimum is found, we represent part of the search space using the contour function (this is possible since our problem involves only 2 parameters) and we superimpose the starting point (in red), the optimisation path (in bleu), and the optimum (in green) to the plot. The history of the optimisation can be retrieved (using neldermead.get) because the '-storehistory' option was set to TRUE.

```
> rosenbrock <- function(x = NULL, index = NULL, fmsfundata = NULL) {
+     return(list(f = 100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2,
+         g = c(), c = c(), gc = c(), index = index, this = list(costfargument = fmsfundata)))
+ }
> x0 <- transpose(c(-1.2, 1))
> nm <- neldermead.new()
> nm <- neldermead.configure(nm, "-numberofvariables", 2)
> nm <- neldermead.configure(nm, "-function", rosenbrock)
> nm <- neldermead.configure(nm, "-x0", x0)
> nm <- neldermead.configure(nm, "-maxiter", 200)
> nm <- neldermead.configure(nm, "-maxfunevals", 300)
> nm <- neldermead.configure(nm, "-tolfunrelative", 10 * .Machine$double.eps)
> nm <- neldermead.configure(nm, "-tolxrelative", 10 * .Machine$double.eps)
> nm <- neldermead.configure(nm, "-simplex0method", "axes")
> nm <- neldermead.configure(nm, "-simplex0length", 1)
> nm <- neldermead.configure(nm, "-method", "variable")
```

```
> nm <- neldermead.configure(nm, "-verbose", 0)
> nm <- neldermead.configure(nm, "-storehistory", TRUE)
> nm <- neldermead.configure(nm, "-verbosetermination", 0)
> nm <- neldermead.search(nm)
> xmin <- ymin <- -2
> xmax <- ymax <- 2
> nx <- ny <- 100
> stepy <- stepx <- (xmax - xmin)/nx
> ydata <- xdata <- seq(xmin, xmax, stepx)
> zdata <- apply(expand.grid(xdata, ydata), 1, function(x) neldermead.function(nm,
+     transpose(x)))
> zdata <- matrix(zdata, ncol = length(ydata))
> optimpath <- matrix(unlist((neldermead.get(nm, "-historyxopt"))),
+     nrow = 2)
> optimpath <- data.frame(x = optimpath[1, ], y = optimpath[2,
+     ])
> contour(xdata, ydata, zdata, levels = c(1, 10, 100, 500,
+     1000, 2000))
> par(new = TRUE, ann = TRUE)
> plot(c(x0[1], optimpath$x[158]), c(x0[2], optimpath$y[158]),
+     col = c("red", "green"), pch = 16, xlab = "x[1]", ylab = "x[2]",
+     xlim = c(xmin, xmax), ylim = c(ymin, ymax))
> par(new = TRUE, ann = FALSE)
> plot(optimpath$x, optimpath$y, col = "blue", type = "l",
+     xlim = c(xmin, xmax), ylim = c(ymin, ymax))
```



Setting the '-verbose' element of the neldermead object to 1 allows to get detailed information about the current optimization process. The following is a sample output for an optimization based on the Nelder and Mead variable-shape simplex algorithm. Only the output corresponding to the iteration #156 is displayed. In order to display specific outputs (or to create specific output files and graphics), the '-outputcommand' option should be used.

```
=================================================================
Iteration \#156 (total = 156)
Function Eval \#298
Xopt: 0.99999999999991 0.999999999999816
Fopt: 8.997809e-27
DeltaFv: 4.492261e-26
Center: 1.00000000000003 1.00000000000007
Size: 4.814034e-13
Vertex \#2/3 : fv=2.649074e-26, x=1.000000e+00 1.000000e+00
Vertex \#3/3 : fv=5.392042e-26, x=1.000000e+00 1.000000e+00
Reflect
xbar=1.00000000000001 1.00000000000003
Function Evaluation \#299 at [0.99999999999996  ]
Function Evaluation \#299 at [0.999999999999907  ]
```

```
xr=[0.99999999999996 0.999999999999907], f(xr)=0.000000
  > Perform reflection
Sort
```

2.3 Example 3: optimization with bound constraints

In the following example, we solve a simple quadratic test case used in Example 1 but in the case where bounds are set for parameter estimates. We begin by defining the cost function, which takes 3 input arguments and returns the value of the objective function as the `f` element of a list. The starting point [1.2 1.9] is used. `neldermead.new` creates a new neldermead object. Then we use `neldermead.configure` to configure the parameters of the problem including the lower `-boundsmin` and upper `-boundsmax` bounds. The initial simplex is computed from `boxnbpoints` random points within the bounds. The variable simplex algorithm by Box is used, which corresponds to the -method 'box' option. `neldermead.search` finally performs the search for the minimum.

```
> quadratic <- function(x = NULL, index = NULL, fmsfundata = NULL) {
+     return(list(f = x[1]^2 + x[2]^2, g = c(), c = c(), gc = c(),
+         index = index, this = list(costfargument = fmsfundata)))
+ }
> set.seed(0)
> x0 <- transpose(c(1.2, 1.9))
> nm <- neldermead.new()
> nm <- neldermead.configure(nm, "-numberofvariables", 2)
> nm <- neldermead.configure(nm, "-function", quadratic)
> nm <- neldermead.configure(nm, "-x0", x0)
> nm <- neldermead.configure(nm, "-verbose", 0)
> nm <- neldermead.configure(nm, "-storehistory", TRUE)
> nm <- neldermead.configure(nm, "-verbosetermination", 0)
> nm <- neldermead.configure(nm, "-method", "box")
> nm <- neldermead.configure(nm, "-boundsmin", c(1, 1))
> nm <- neldermead.configure(nm, "-boundsmax", c(2, 2))
> nm <- neldermead.search(nm)
> transpose(neldermead.get(nm, "-xopt"))

         [,1]     [,2]
[1,] 1.000001 1.000001
```

2.4 Example 4: optimization with nonlinear inequality constraints

In the following example, we solve Michalewicz's G_6 test problem using Box's methods [4] [1]. This problem consists in minimizing: $G_6(x) = (x_1 - 10)^3 + (x_2 - 20)^3$, given the nonlinear constraints:

$c1: \quad (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0$

$c2: \quad -(x_1 - 6)^2 - (x_2 - 5)^2 + 82.81 \geq 0$

and bounds: $13 \leq x_1 \leq 100$, $0 \leq x_2 \leq 100$.

We begin by defining the `michalewicz` function, which takes 3 input arguments and return the value of the objective function and the constraint evaluations as the `f` and `c` elements of a list. `neldermead.new` creates a new neldermead object. Then we use `neldermead.configure` to configure the parameters of the problem, including the lower `-boundsmin` and upper `-boundsmax`

[1] Example suggested by Pascal Grandeau

bounds. The initial simplex is computed from `boxnbpoints` random points within the bounds. The variable simplex algorithm by Box is used, which corresponds to the -method 'box' option. `neldermead.search` finally performs the search for the minimum. The starting point ([15 4.99]) like all the vertices of the optimization simplex must be feasible, i.e. they must satisfy all constraints and bounds. Constraints are enforced by ensuring that all arguments of `c` in the cost function output are positive or null. Note that the boundaries were set to stricter ranges to limit the sensitivity of the solution to the initial guesses.

```
> michalewicz <- function(x = NULL, index = NULL, fmsfundata = NULL) {
+     f <- c()
+     c <- c()
+     if (index == 2 | index == 6)
+         f <- (x[1] - 10)^3 + (x[2] - 20)^3
+     if (index == 5 | index == 6)
+         c <- c((x[1] - 5)^2 + (x[2] - 5)^2 - 100, 82.81 -
+             ((x[1] - 6)^2 + (x[2] - 5)^2))
+     varargout <- list(f = f, g = c(), c = c, gc = c(), index = index,
+         this = list(costfargument = fmsfundata))
+     return(varargout)
+ }
> set.seed(0)
> x0 <- transpose(c(15, 4.99))
> nm <- neldermead.new()
> nm <- neldermead.configure(nm, "-numberofvariables", 2)
> nm <- neldermead.configure(nm, "-nbineqconst", 2)
> nm <- neldermead.configure(nm, "-function", michalewicz)
> nm <- neldermead.configure(nm, "-x0", x0)
> nm <- neldermead.configure(nm, "-maxiter", 300)
> nm <- neldermead.configure(nm, "-maxfunevals", 1000)
> nm <- neldermead.configure(nm, "-simplex0method", "randbounds")
> nm <- neldermead.configure(nm, "-boxnbpoints", 3)
> nm <- neldermead.configure(nm, "-storehistory", TRUE)
> nm <- neldermead.configure(nm, "-method", "box")
> nm <- neldermead.configure(nm, "-boundsmin", c(13, 0))
> nm <- neldermead.configure(nm, "-boundsmax", c(20, 10))
> nm <- neldermead.search(nm)
> transpose(neldermead.get(nm, "-xopt"))

       [,1]      [,2]
[1,] 14.095 0.8429608

> neldermead.get(nm, "-fopt")

[1] -6961.814
```

2.5 Example 5: Passing data to the cost function

In the following example, we use a simple example to illustrate how to pass user-defined arguments to a user-defined cost function. We try to find the mean and standard deviation of some normally distributed data using maximum likelihood (actually a modifed negative log-likelihood approach) [2].

[2] Example suggested by Mark Taper

We begin by defining the `negLL` function, which takes 3 input arguments and return the value of the objective function. The random dataset is then generated and stored in the list `fmsdundata`. `neldermead.new` creates a new neldermead object. Then we use `neldermead.configure` to configure the parameters of the problem, including `costfargument`, set to `fmsdundata`, and the lower `-boundsmin` and upper `-boundsmax` bounds (the standard deviations has to be positive). The variable simplex algorithm by Box is used. `neldermead.search` finally performs the search for the minimum.

```
> negLL <- function(x = NULL, index = NULL, fmsfundata = NULL) {
+     mn <- x[1]
+     sdv <- x[2]
+     out <- -sum(dnorm(fmsfundata$data, mean = mn, sd = sdv,
+         log = TRUE))
+     return(list(f = out, index = index, this = list(costfargument = fmsfundata)))
+ }
> set.seed(12345)
> fmsfundata <- list(data = rnorm(500, mean = 50, sd = 2))
> attr(fmsfundata, "type") <- "T_FARGS"
> x0 <- transpose(c(45, 3))
> nm <- neldermead.new()
> nm <- neldermead.configure(nm, "-numberofvariables", 2)
> nm <- neldermead.configure(nm, "-function", negLL)
> nm <- neldermead.configure(nm, "-x0", x0)
> nm <- neldermead.configure(nm, "-costfargument", fmsfundata)
> nm <- neldermead.configure(nm, "-maxiter", 500)
> nm <- neldermead.configure(nm, "-maxfunevals", 1500)
> nm <- neldermead.configure(nm, "-method", "box")
> nm <- neldermead.configure(nm, "-storehistory", TRUE)
> nm <- neldermead.configure(nm, "-boundsmin", c(-100, 0))
> nm <- neldermead.configure(nm, "-boundsmax", c(100, 100))
> nm <- neldermead.search(this = nm)
> transpose(neldermead.get(nm, "-xopt"))

         [,1]     [,2]
[1,] 50.16492 1.978316

> neldermead.get(nm, "-fopt")

[1] 1050.592
```

3 fminsearch

`fminsearch` function is based on a specialized use of the more general `neldermead` function bundle and searches for the unconstrained minimum of a given cost function. This function corresponds to the Matlab (or Scilab) fminsearch function. Additional information and examples are available in `?fminsearch` from a R environment.

4 References

[1] C.T. Kelley. *Iterative Methods for Optimization.* SIAM Frontiers in Applied Mathematics, Philadelphia, PA, 1999.

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

[3] J.A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[4] Z. Michalewicz and D.B. Fogel. *How to solve it: modern heuristics*, chapter Constraint-handling techniques, pages 231–270. 2004.

[5] M.J. Box. A New Method of Constrained Optimization and a Comparison With Other Methods. *The Computer Journal*, 1(8):42–52, 1965.

[6] R. O'Neill. Algorithm AS47 - Function minimization using a simplex procedure. *Applied Statistics*, 20:338–345, 1971.

[7] W. Spendley and G.R. Hext and F.R. Himsworth. Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation. *Technometrics*, 4:441–461, 1962.

5 Dependencies of fminsearch

We illustrate in the figures below the network of functions of the **neldermead**, **optimbase**, and **optimsimplex** packages that are called from the `fminsearch` functions. This large network is broken down in 6 plots, which are shown in the order functions are called. Green boxes represent functions that are not expanded on a given plot but on a previous or later one.

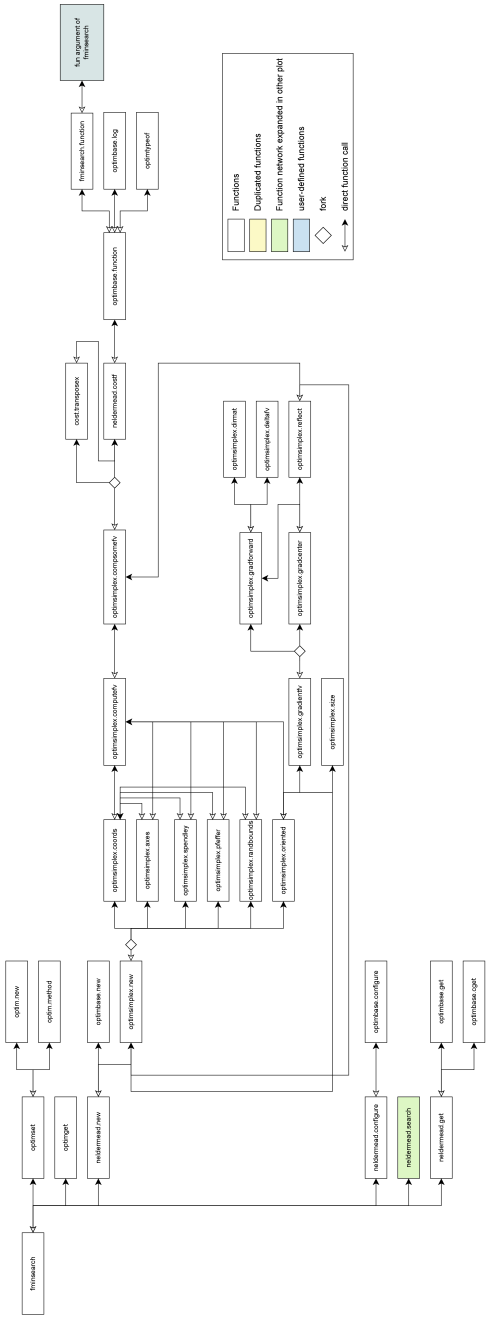


Figure 1: fminsearch function network (1/6)

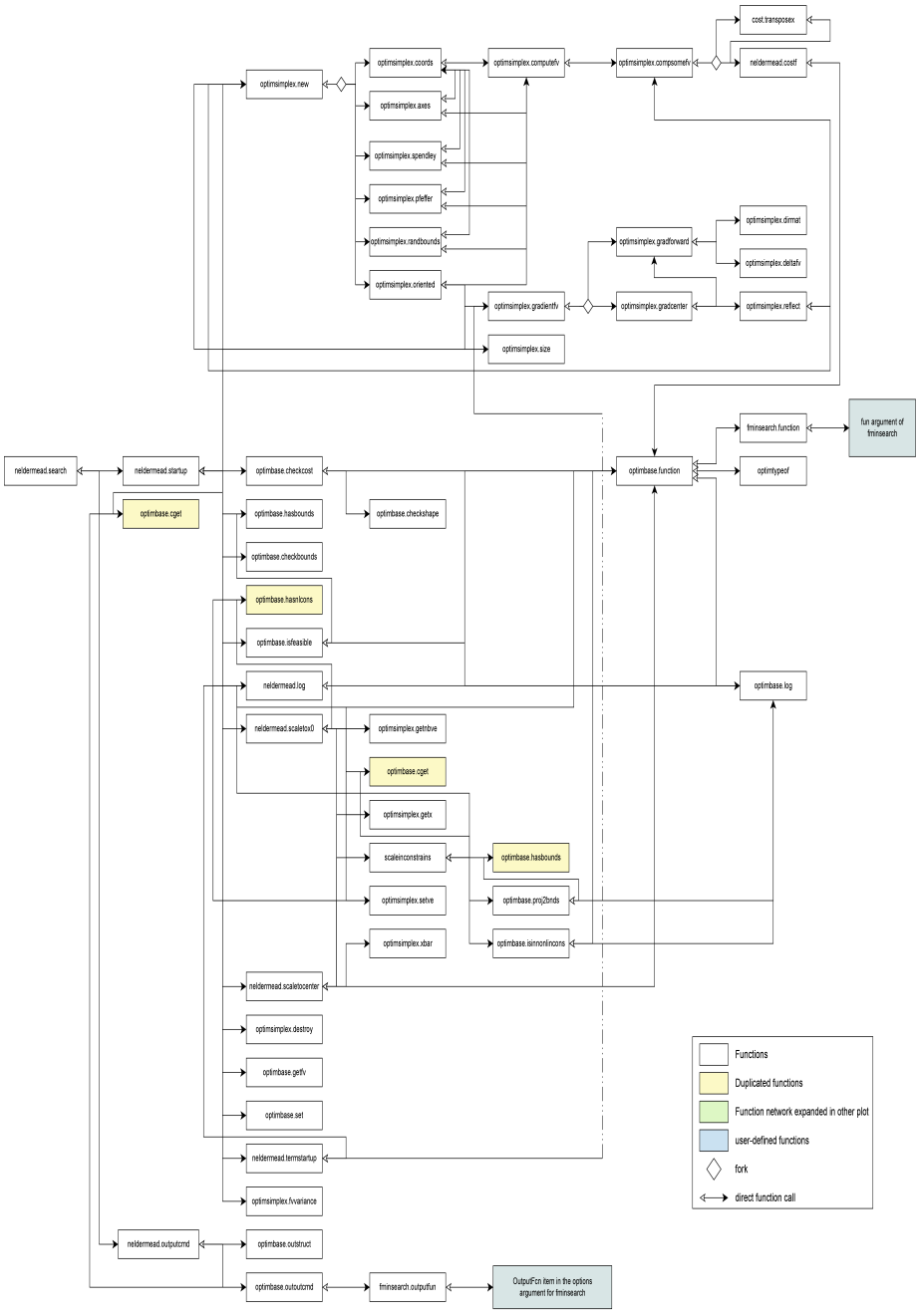


Figure 2: fminsearch function network (2/6)

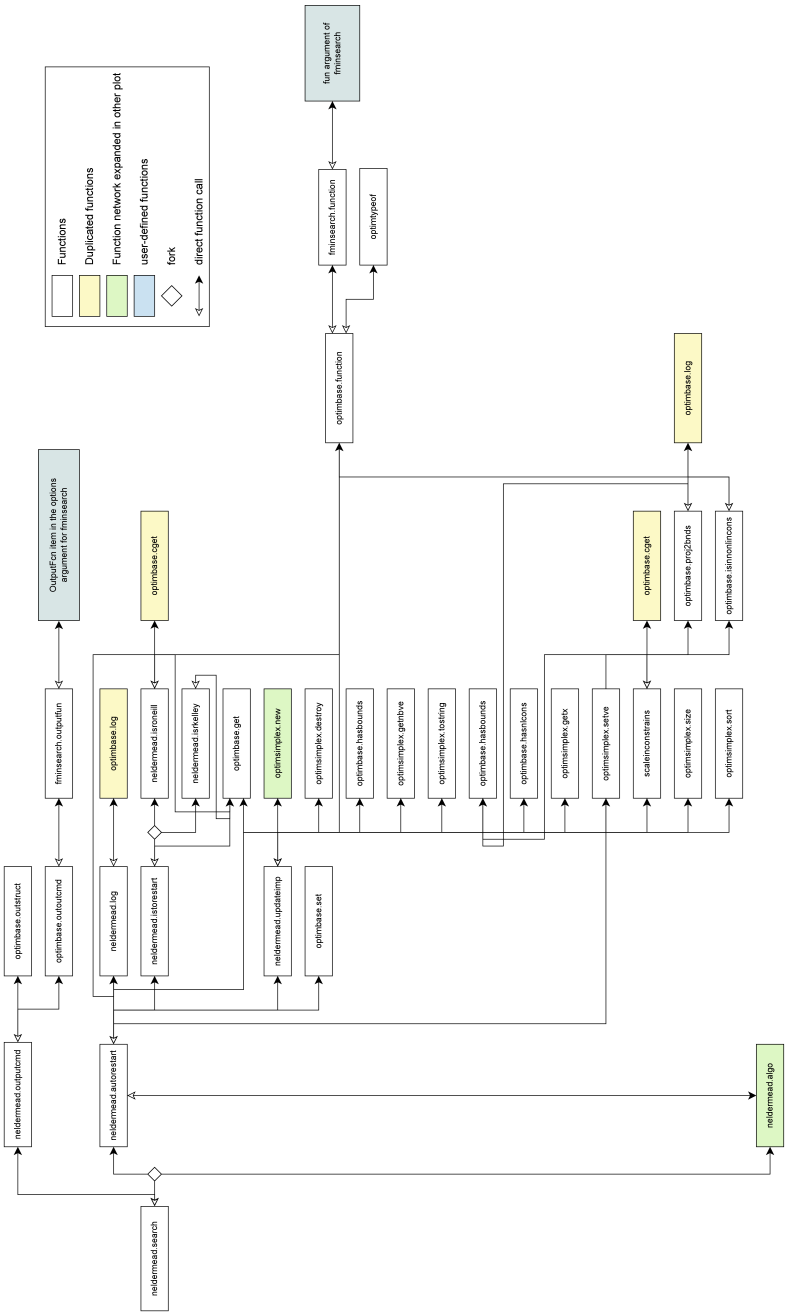


Figure 3: fminsearch function network (3/6)

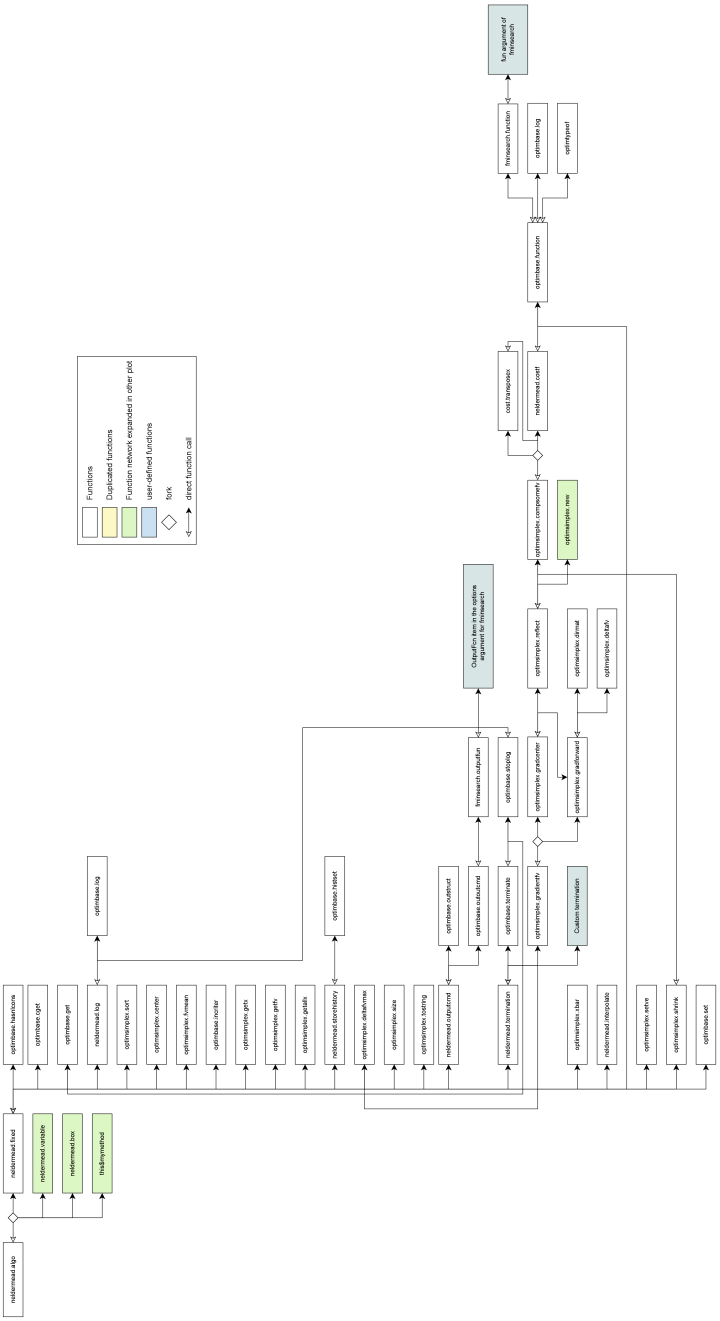


Figure 4: fminsearch function network (4/6)

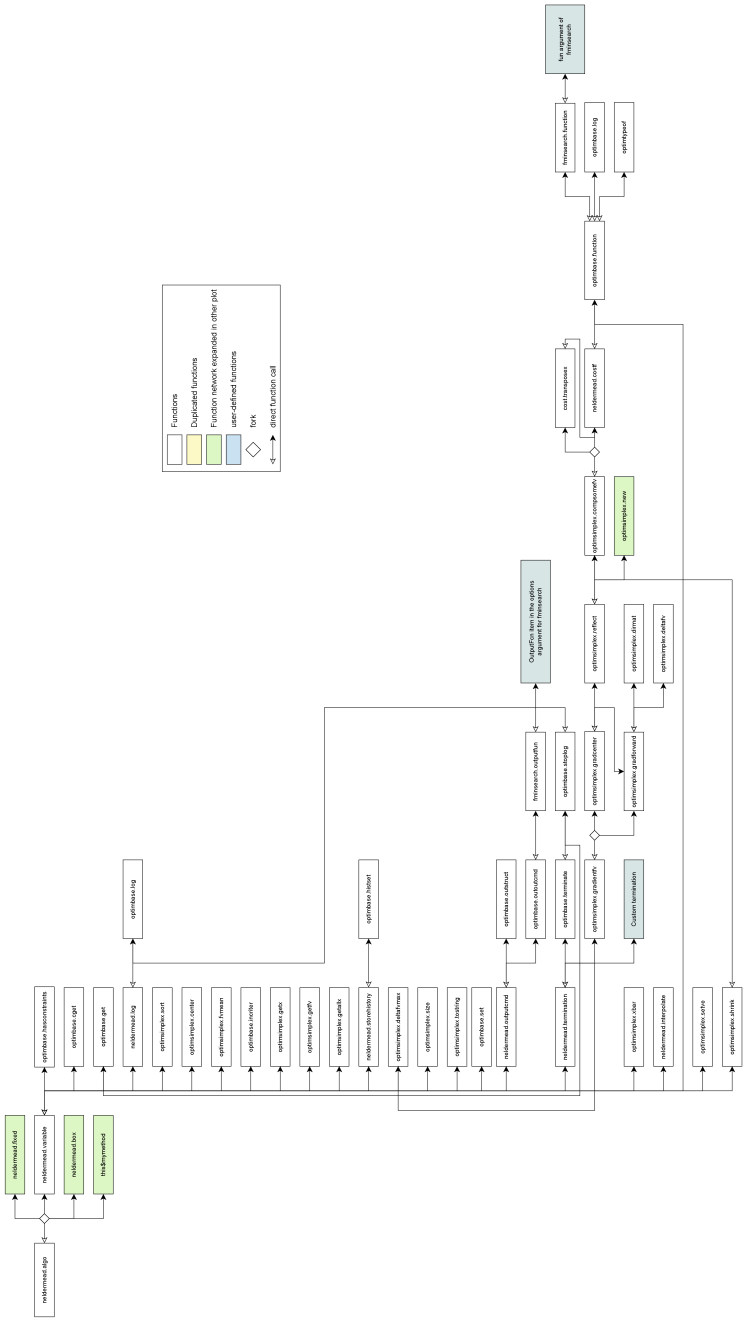


Figure 5: fminsearch function network (5/6)

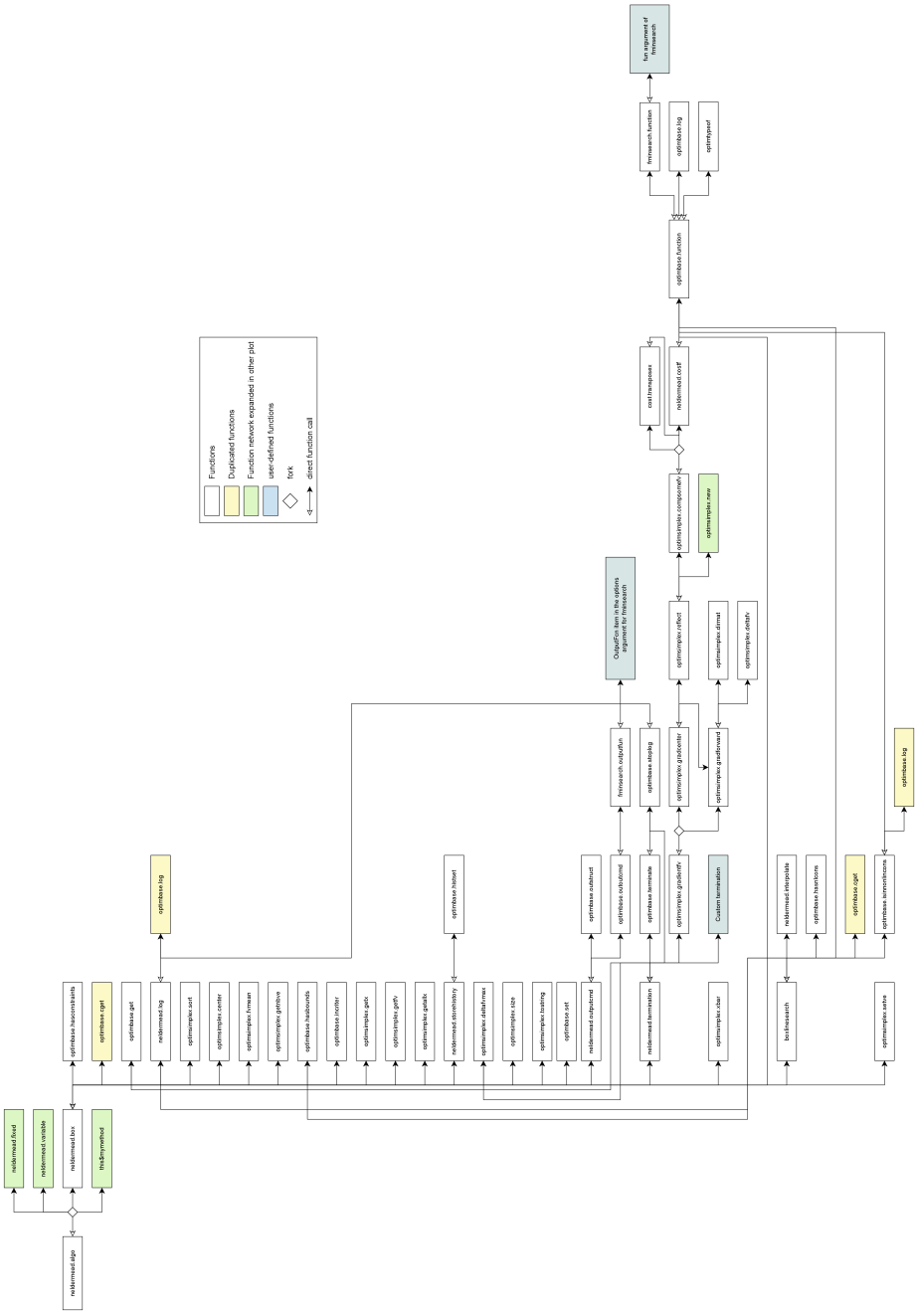


Figure 6: fminsearch function network (6/6)

6 Help on neldermead functions

`neldermead-package` *R port of the Scilab neldermead module*

Description

The goal of this package is to provide a Nelder-Mead direct search optimization method. That 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.

Design

This package provides 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 simplified Nelder-Mead algorithm. Specific termination criteria, initial simplex and auxiliary settings are automatically configured.
- `optimset`, `optimget` provide commands to emulate their Scilab counterparts.
- `optimplotfunccount`, `optimplotx` and `optimplotfval` provide plotting features for the fminsearch function (Not implemented yet).
- `nmplot` provides a high-level component which provides directly output pictures for Nelder-Mead algorithm. (Not implemented yet).

The current component is based on the following packages

- **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...),

Features

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

- Provides 3 algorithms, including
  - the fixed shape algorithm of Spendley et al.,
  - the variable shape algorithm of Nelder and Mead,
  - Box's 'complex' algorithm managing bounds and nonlinear inequality constraints based on arbitrary number of vertices in the simplex.

- 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 formula of Spendley et al.,
  - 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 Box 1965),
  - 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 (Not yet implemented).
- 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...
- Kelley restart based on simplex gradient.
- O'Neill restart based on factorial search around optimum.

Details

Package:	neldermead
Type:	Package
Version:	1.0-4
Date:	2010-10-15
License:	CeCILL-2
LazyLoad:	yes

See `vignette('neldermead',package='neldermead')` for more information.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (`<sb.pmlab@gmail.com>`)

References

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

'A Simplex Method for Function Minimization', Nelder, J. A. and Mead, R., The Computer Journal, 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 by British Computer Society

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

'Detection and Remediation of Stagnation in the Nelder–Mead Algorithm Using a Sufficient Decrease Condition', Kelley C. T., SIAM J. on Optimization, 1999

'Iterative Methods for Optimization', C. T. Kelley, SIAM Frontiers in Applied Mathematics, 1999

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

See Also

`optimbase` `optimsimplex`

`costf.transposex` *Cost Function Call*

Description

Call the cost function after transposition of the value of the point estimate x, so that the input row vector, given by optimsimplex, is transposed into a column vector as required by the cost function.

Usage

```
costf.transposex(x = NULL, this = NULL)
```

Arguments

`x`	The point estimate provide as a row matrix.
`this`	A neldermead object.

Value

Return the value of the cost function (called by `neldermead.costf`.)

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`neldermead.costf`

`fmin.gridsearch` *Grid evaluation of an unconstrained cost function*

Description

Evaluate an unconstrained cost function on a grid of points around a given initial point estimate.

Usage

```
fmin.gridsearch(fun = NULL, x0 = NULL, xmin = NULL,
                      xmax = NULL, npts = 3, alpha = 10)
```

Arguments

`fun`	An unconstrained cost function, similar to those used in the `fminsearch` function.
`x0`	The initial point estimate, provided as a numeric vector.
`xmin`	Optional: a vector of lower bounds.
`xmax`	Optional: a vector of upper bounds.
`npts`	A integer scalar greater than 2, indicating the number of evaluation points will be used on each dimension to build the search grid.
`alpha`	A vector of positive numbers, which give the factor(s) used to calculate the evaluation range of each dimension of the search grid (see Details). If `alpha` length is lower than that of `x0`, elements of `alpha` are recycled. If its length is higher than that of `x0`, `alpha` is truncated.

Details

`fmin.gridsearch` evaluates the cost function at each point of a grid of `npts^length(x0)` points. If lower (`xmin`) and upper (`xmax`) bounds are provided, the range of evaluation points is limited by those bounds and `alpha` is not used. Otherwise, the range of evaluation points is defined as `[x0/alpha,x0*alpha]`.

The actual evaluation of the cost function is delegated to `optimbase.gridsearch`.

Value

Return a data.frame with the coordinates of the evaluation point, the value of the cost function and its feasibility. Because the cost function is unconstrained, it is always feasible. The data.frame is ordered by feasibility and increasing value of the cost function.

Author(s)

Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

fminsearch, optimbase.gridsearch

---

fminsearch.function *fminsearch Cost Function Call*

Description

This function calls the cost function and makes it match neldermead requirements. It is used in the fminsearch function as the function element of the neldermead object (see ?neldermead.new and ?neldermead.configure).

Usage

```
fminsearch.function(x = NULL, index = NULL, fmsfundata = NULL)
```

Arguments

x	A single column vector of parameter estimates.
index	An integer variable set to 2, indicating that only the cost function is to be computed by the algorithm.
fmsfundata	A list with a type attribute set to 'T_FARGS' and with (at least) a fun element, which contain the user-defined cost function.

Value

Returns a list with the following elements:

f The value of the cost function at the current point estimate.

index The same index variable.

this A list with a single element costargument which contains fmsfundata.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

fminsearch, neldermead.new, neldermead.configure,

`fminsearch.outputfun` *fminsearch Output Function Call*

Description

This function calls the output function and make it match neldermead requirements. It is used in the `fminsearch` function as the `outputcommand` element of the neldermead object (see `?neldermead.new` and `?neldermead.configure`).

Usage

```
fminsearch.outputfun(state = NULL, data = NULL, fmsdata = NULL)
```

Arguments

`state`	The current state of the algorithm either 'init', 'iter' or 'done'.
`data`	The data at the current state. This is a list with a 'type' attribute set to 'T_NMDATA' and with the following elements: **x** The current parameter estimates. **fval** The current value of the cost function. **simplex** The current simplex object. **iteration** The number of iterations performed. **funccount** The number of function evaluations. **step** The type of step in the previous iteration.
`fmsdata`	This is a list with a 'type' attribute set to 'T_FARGS' which contains specific data of the `fminsearch` algorithm: **Display** what to display **OutputFcn** the array of output functions **PlotFcns** the array of plot functions

Value

This function does not return any data, but execute the output function(s).

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (`<sb.pmlab@gmail.com>`)

See Also

`fminsearch`, `neldermead.new`, `neldermead.configure`,

`fminsearch` *Computation of the unconstrained minimimum of given function with the Nelder-Mead algorithm.*

Description

This function searches for the unconstrained minimum of a given cost function. The provided algorithm is a direct search algorithm, i.e. an algorithm which does not use the derivative of the cost function. It is based on the update of a simplex, which is a set of k>=n+1 vertices, where each vertex is associated with one point and one function value. This algorithm is the Nelder-Mead algorithm. This function is based on a specialized use of the more general `neldermead` function bundle. Users who want to have a more flexible solution based on direct search algorithms should consider using the `neldermead` functions instead of the `fminsearch` function.

Usage

```
fminsearch(fun = NULL, x0 = NULL, options = NULL)
```

Arguments

`fun` A cost function.

`x0` A numerical vector of initial guesses (length n).

`options` A list of optimization options, which drives the behaviour of `fminsearch`. These options must be set with the `optimset` function (see `?optimset`) which returns a list with the following elements:

MaxIter The maximum number of iterations. The default is 200 * n.

MaxFunEvals The maximum number of evaluations of the cost function. The default is 200 * n.

TolFun The absolute tolerance on function value. The default value is 1.e-4.

TolX The absolute tolerance on simplex size. The default value is 1.e-4.

Display The verbose level.

OutputFcn The output function, or a list of output functions called at the end of each iteration. The default value is NULL.

PlotFcns The plot function, or a list of plotput functions called at the end of each iteration. The default value is empty.

Details

Termination criteria

In this section, we describe the termination criteria used by fminsearch. The criteria is based on the following variables:

ssize the current simplex size,

shiftfv the absolute value of the difference of function value between the highest and lowest vertices.

If both `ssize < options$TolX` and `shiftfv < options$TolFun` conditions are true, then the iterations stop. The size of the simplex is computed using the 'sigmaplus' method of the **optimsimplex** package. The 'sigmamplus' size is the maximum length of the vector from each vertex to the first vertex. It requires one loop over the vertices of the simplex.

The initial simplex

The `fminsearch` algorithm uses a special initial simplex, which is an heuristic depending on the initial guess. The strategy chosen by `fminsearch` corresponds to the content of `simplex0method` element of the neldermead pbject (set to 'pfeffer'). It is applied using the content of the `simplex0deltausual` (0.05) and `simplex0deltazero` (0.0075) elements. Pfeffer's method is an heuristic which is presented in 'Global Optimization Of Lennard-Jones Atomic Clusters' by Ellen Fan. It is due to L. Pfeffer at Stanford. See in the help of optimsimplex for more details.

The number of iterations

In this section, we present the default values for the number of iterations in `fminsearch`.

The `options` input argument is an optionnal list which can contain the `MaxIter` field, which stores the maximum number of iterations. The default value is 200n, where n is the number of variables. The factor 200 has not been chosen by chance, but is the result of experiments performed against quadratic functions with increasing space dimension. This result is presented in 'Effect of dimensionality on the nelder-mead simplex method' by Lixing Han and Michael Neumann. This paper is based on Lixing Han's PhD, 'Algorithms in Unconstrained Optimization'. The study is based on numerical experiments with a quadratic function where the number of terms depends on the dimension of the space (i.e. the number of variables). Their study showed that the number of iterations required to reach the tolerance criteria is roughly 100n. Most iterations are based on inside contractions. Since each step of the Nelder-Mead algorithm only require one or two function evaluations, the number of required function evaluations in this experiment is also roughly 100n.

Output and plot functions

The `optimset` function can be used to configure one or more output and plot functions. The output or plot function is expected to have the following definition:

```
myfun <- function(x , optimValues , state)
```

The input arguments `x`, `optimValues` and `state` are described in detail in the `optimset` help page. The `optimValues$procedure` field represents the type of step preformed at the current iteration and can be equal to one of the following strings:

- '' (the empty string),
- 'initial simplex',
- 'expand',
- 'reflect',
- 'contract inside',
- 'contract outside'.

Value

Return a list with the following fields:

x The vector of n numeric values, minimizing the cost function.

fval The minimum value of the cost function.

exitflag The flag associated with exist status of the algorithm. The following values are available:

-1 The maximum number of iterations has been reached.

0 The maximum number of function evaluations has been reached.

1 The tolerance on the simplex size and function value delta has been reached. This signifies that the algorithm has converged, probably to a solution of the problem.

output A list which stores detailed information about the exit of the algorithm. This list contains the following fields:

algorithm A string containing the definition of the algorithm used, i.e. 'Nelder-Mead simplex direct search'.

funcCount The number of function evaluations.

iterations The number of iterations.

message A string containing a termination message.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

References

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

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

'Iterative Methods for Optimization', C. T. Kelley, SIAM Frontiers in Applied Mathematics, 1999

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

'Effect of dimensionality on the nelder-mead simplex method', Lixing Han and Michael Neumann, Optimization Methods and Software, 21, 1, 1–16, 2006.

'Algorithms in Unconstrained Optimization', Lixing Han, Ph.D., The University of Connecticut, 2000.

'Global Optimization Of Lennard-Jones Atomic Clusters' Ellen Fan, Thesis, February 26, 2002, McMaster University

See Also

`optimset`

Examples

```
#In the following example, we use the fminsearch function to compute the minimum
#of the Rosenbrock function. We first define the function 'banana', and then use
#the fminsearch function to search the minimum, starting with the initial guess
#(-1.2, 1.0). In this particular case, 85 iterations are performed with 159
#function evaluations
  banana <- function(x){
    y <- 100*(x[2]-x[1]^2)^2 + (1-x[1])^2
  }
  sol <- fminsearch(banana, c(-1.2,1))
  sol

#In the following example, we configure the absolute tolerance on the size of
#the simplex to a larger value, so that the algorithm performs less iterations.
#Since the default value of 'TolX' for the fminsearch function is 1.e-4, we
#decide to use 1.e-2. The optimset function is used to create an optimization
#option list and the field 'TolX' is set to 1.e-2. The options list is then
#passed to the fminsearch function as the third input argument. In this
#particular case, the number of iterations is 70 with 130 function evaluations.

  opt <- optimset(TolX=1.e-2)
  sol <- fminsearch(banana, c(-1.2,1), opt)
  sol

#In the following example, we want to produce intermediate outputs of the
#algorithm. We define the outfun function, which takes the current point x as
#input argument. The function plots the current point into the current graphic
#window with the plot function. We use the 'OutputFcn' feature of the optimset
#function and set it to the output function. Then the option list is passed
#to the fminsearch function. At each iteration, the output function is called
#back, which creates and update a plot. While this example creates a 2D plot,
#the user may customized the output function so that it writes a message in
#the console, write some data into a data file, etc... The user can distinguish
#between the output function (associated with the 'OutputFcn' option) and the
#plot function (associated with the 'PlotFcns' option). See the optimset for
#more details on this feature.

  outfun <- function(x, optimValues, state){
    plot(x[1],x[2],xlim=c(-1.5,1.5),ylim=c(-1.5,1.5))
    par(new=TRUE)
  }
  opt <- optimset(OutputFcn=outfun)
  sol <- fminsearch(banana, c(-1.2,1), opt)
  sol

#The 'Display' option allows to get some input about the intermediate steps of
#the algorithm as well as to be warned in case of a convergence problem.
#In the following example, we present what happens in case of a convergence
#problem. We set the number of iterations to 10, instead of the default 400
#iterations. We know that 85 iterations are required to reach the convergence
#criteria. Therefore, the convergence criteria is not met and the maximum number
#of iterations is reached.
```

```
  opt <- optimset(MaxIter=10)
  sol <- fminsearch(banana, c(-1.2,1), opt)

#Since the default value of the 'Display' option is 'notify', a message is
#generated, which warns the user about a possible convergence problem. The
#previous script produces the following output.
# Exiting: Maximum number of iterations has been exceeded
#          - increase MaxIter option.
#          Current function value: 4.1355598

#In the following example, we present how to display intermediate steps used by
#the algorithm. We simply set the 'Display' option to the 'iter' value. This
#option allows to see the number of function evaluations, the minimum function
#value and which type of simplex step is used for the iteration.
  opt <- optimset(Display='iter')
  sol <- fminsearch(banana, c(-1.2,1), opt)
  sol
```

`neldermead.algo` *Nelder-Mead Algorithm*

Description

`neldermead.algo` performs an optimization without restart using the method associated with the `method` element of the neldermead object; `neldermead.fixed`, `neldermead.variable`, `neldermead.box`, `boxlinesearch`, `neldermead.storehistory`, `neldermead.termination`, and `neldermead.interpolate` are utility functions for `neldermead.algo`.

Usage

```
  neldermead.algo(this = NULL)
  neldermead.fixed(this = NULL)
  neldermead.variable(this = NULL)
  neldermead.box(this = this)
  boxlinesearch(this = NULL, n = NULL, xbar = NULL, xhigh = NULL, fhigh = NULL,
                rho = NULL)
  neldermead.storehistory(this = NULL, n = NULL, fopt = NULL, xopt = NULL,
                          xcoords = NULL)
  neldermead.termination(this = NULL, fvinitial = NULL, oldfvmean = NULL,
                         newfvmean = NULL, previousxopt = NULL,
                         currentxopt = NULL, simplex = NULL)
  neldermead.interpolate(x1 = NULL, x2 = NULL, fac = NULL)
```

Arguments

`this`	A neldermead object.
`n`	Number of variables.

`xbar`	The centroid.
`xhigh`	The high point.
`fhigh`	The value of the cost function at `xhigh`.
`rho`	The reflection factor.
`fopt`	The current value of the function at the current optimum point estimate.
`xopt`	The current optimum point estimate.
`xcoords`	Matrix of size n x n+1, coordinates of the n+1 vertices
`fvinitial`	The initial cost function value.
`oldfvmean`	The old cost function value average on the simplex.
`newfvmean`	The new cost function value average on the simplex.
`previousxopt`	The previous point estimate.
`currentxopt`	The current point estimate.
`simplex`	The simplex. The best point estimate in the simplex is expected to be stored at 1, while the worst point estimate in the simplex is expected to be stored at n+1.
`x1`	The first reference point estimate to perform the interpolation.
`x2`	The second reference point estimate to perform the interpolation.
`fac`	A factor to perform the interpolation.

Details

`neldermead.fixed` The simplex algorithm with fixed size simplex. We implement the following 'rules' of the method of Spendley et al.

- Rule 1 is strictly applied, but the reflection is done by reflection of the high point, since we minimize a function instead of maximizing it, like Spendley.
- Rule 2 is NOT implemented, as we expect that the function evaluation is not subject to errors.
- Rule 3 is applied, i.e. reflection with respect to next to high point. A shrink step is included, with shrinkage factor sigma.

Rule 1. Ascertain the lowest reading y, of yi ... Yk+1 Complete a new simplex Sp by excluding the point Vp corresponding to y, and replacing it by V* defined as above.

Rule 2. If a result has occurred in (k + 1) successive simplexes, and is not then eliminated by application of Rule 1, do not move in the direction indicated by Rule 1, or at all, but discard the result and replace it by a new observation at the same point.

Rule 3. If y is the lowest reading in So , and if the next observation made, y* , is the lowest reading in the new simplex S , do not apply Rule 1 and return to So from Sp . Move out of S, by rejecting the second lowest reading (which is also the second lowest reading in So).

`neldermead.variable` The original Nelder-Mead algorithm, with variable-size simplex.

`neldermead.box` The Nelder-Mead algorithm, with variable-size simplex and modifications by Box for bounds and inequality constraints.

`boxlinesearch` Called by `neldermead.box`, i.e. Box's method. Perform a line search from xbar, on the line (xhigh,xbar). The reflected point estimate satisfies the following constraints:

- fr < fhigh
- xr satisfies the bounds constraints
- xr satisfies the nonlinear positive inequality constraints
- xr satisfies the linear positive inequality constraints

The method is based on projection and scaling toward the centroid.

`neldermead.storehistory` Store the optimization history into the neldermead object.

`neldermead.termination` Determine if the algorithm must continue or terminate. The function uses the cost function average in the simplex instead of the best cost function value. This is because the function average changes at each iteration. Instead, the best function value has a step-by-step evolution and may not change between two successive iterations, leading to a stop of the algorithm.

`neldermead.interpolate` Compute the point estimate xi as an interpolation between x1 and x2, as follows: xi = (1+fac)x1 - fac*x2

Value

`neldermead.fixed`, `neldermead.variable`, **and** `neldermead.box` Return the updated neldermead object, containing the optimum point estimate.

`boxlinesearch` Return a list with the following elements:

this The updated neldermead object.

status TRUE if the search is successful, FALSE otherwise.

xr The reflected point estimate.

fr The value of the cost function at `xr`.

`neldermead.storehistory` Return the updated neldermead object.

`neldermead.termination` Return a list with the following elements:

this The updated neldermead object

terminate TRUE if the algorithm terminates, FALSE if the algorithm must continue.

status The termination status: 'continue', 'maxiter', 'maxfuneval', 'tolf', 'tolx', 'tolsize', 'tolsizedeltafv', 'kelleystagnation', 'tolboxf', 'tolvariance' or the user-defined termination status.

`neldermead.interpolate` Return a new point estimate, i.e. a column vector.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (`<sb.pmlab@gmail.com>`)

`neldermead.configure` *Neldermead Object Configuration*

Description

Configure the current neldermead object with the given value for the given key.

Usage

```
neldermead.configure(this = NULL, key = NULL, value = NULL)
```

Arguments

`this`	The current neldermead object.
`key`	The key to configure. See details for the list of possible keys.
`value`	The value to assign to the key.

Details

`neldermead.configure` sets the content of the `key` element of the neldermead object `this` to `value`. If `key` is a sub-element of `this$optbase`, `value` is assigned by `optimbase.configure`.

The main available keys are the following:

'-verbose' Set to 1 to enable verbose logging.

'-verbosetermination' Set to 1 to enable verbose termination logging.

'-x0' The initial guess, as a n x 1 column vector, where n is the number of variables.

'-maxfunevals' The maximum number of function evaluations. If this criteria is triggered during optimization, the status of the optimization is set to 'maxfuneval'.

'-maxiter' The maximum number of iterations. If this criteria is triggered during optimization, the status of the optimization is set to 'maxiter'.option

'-tolfunabsolute' The absolute tolerance for the function value.

'-tolfunrelative' The relative tolerance for the function value.

'-tolfunmethod' The method used for the tolerance on function value in the termination criteria. The following values are available: TRUE, FALSE. If this criteria is triggered, the status of the optimization is set to 'tolf'.

'-tolxabsolute' The absolute tolerance on x.

'-tolxrelative' The relative tolerance on x.

'-tolxmethod' The method used for the tolerance on x in the termination criteria. The following values are available: TRUE, FALSE. If this criteria is triggered during optimization, the status of the optimization is set to 'tolx'.

'-function' The objective function, which computes the value of the cost and the non linear constraints, if any. See `vignette('neldermead',package='neldermead')` for the details of the communication between the optimization system and the cost function.

'-costfargument' An additionnal argument, passed to the cost function.

'-outputcommand' A command which is called back for output. See `vignette('neldermead',package='neld` for the details of the communication between the optimization system and the output command function.

'-outputcommandarg' An additionnal argument, passed to the output command.option

'-numberofvariables' The number of variables to optimize.

'-storehistory' Set to TRUE to enable the history storing.

'-boundsmin' The minimum bounds for the parameters.

'-boundsmax' The maximum bounds for the parameters.

'-nbineqconst' The number of inequality constraints.

'-method' The name of the algorithm to use. The following methods are available:

- **'fixed'** the fixed simplex shape algorithm of Spendley et al. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)
- **'variable'** the variable simplex shape algorithm of Nelder and Mead. This algorithm is for unconstrained problems (i.e. bounds and non linear constraints are not taken into account)
- **'box'** Box's complex algorithm. This algorithm takes into account bounds and nonlinear inequality constraints.
- **'mine'** the user-defined algorithm, associated with the `mymethod` element. See `vignette('neldermead',pa` for details.

'-simplex0method' The method to use to compute the initial simplex. The first vertex in the simplex is always the initial guess associated with the `x0` element. The following methods are available:

- **'given'** The coordinates associated with the `coords0` element are used to compute the initial simplex, with arbitrary number of vertices. This allows the user to setup the initial simplex by a specific method which is not provided by the current package (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm is to be reproduced (for example the Mac Kinnon test case). The given matrix is expected to have nbve rows and n columns, where n is the dimension of the problem and nbve is the number of vertices.
- **'axes'** The simplex is computed from the coordinate axes and the length associated with the `simplex0length` element.
- **'spendley'** The simplex is computed so that it is regular with the length associated with the `simplex0length` element (i.e. all the edges have the same length).
- **'pfeffer'** The simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the -simplex0deltausual and -simplex0deltazero.
- **'randbounds'** The simplex is computed from the bounds and a random number. This option is available only if bounds are available: if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the `boxnbpoints` element.

'-coords0' The coordinates of the vertices of the initial simplex. If the `simplex0method` element is set to 'given', these coordinates are used to compute the initial simplex. This matrix is expected to have shape nbve x n, where nbve is the number of vertices and n is the number of variables.

'-simplex0length' The length to use when the initial simplex is computed with the 'axes' or 'spendley' methods. If the initial simplex is computed from 'spendley' method, the length is expected to be a scalar value. If the initial simplex is computed from 'axes' method, it may be either a scalar value or a vector of values, of length n, where n is the number of variables.

'-simplex0deltausual' The relative delta for non-zero parameters in 'pfeffer' method.

'-simplex0deltazero' The absolute delta for non-zero parameters in 'pfeffer' method.

'-rho' The reflection coefficient. This parameter is used when the `method` element is set to 'fixed' or 'variable'.

'-chi' The expansion coefficient. This parameter is used when the `method` element is set to 'variable'.

'-gamma' The contraction coefficient. This parameter is used when the `method` element is set to 'variable'.

'-sigma' The shrinkage coefficient. This parameter is used when the `method` element is set to 'fixed' or 'variable'.

'-tolsimplexizemethod' Set to FALSE to disable the tolerance on the simplex size. If this criteria is triggered, the status of the optimization is set to 'tolsize'. When this criteria is enabled, the values of the `tolsimplexizeabsolute` and `tolsimplexizerelative` elements are used in the termination criteria. The method to compute the size is the 'sigmaplus' method.

'-tolsimplexizeabsolute' The absolute tolerance on the simplex size.

'-tolsimplexizerelative' The relative tolerance on the simplex size.

'-tolssizedeltafvmethod' Set to TRUE to enable the termination criteria based on the size of the simplex and the difference of function value in the simplex. If this criteria is triggered, the status of the optimization is set to 'tolsizedeltafv'. This termination criteria uses the values of the `tolsimplexizeabsolute` and `toldeltafv` elements.option

'-toldeltafv' The absolute tolerance on the difference between the highest and the lowest function values.

'-tolvarianceflag' Set to TRUE to enable the termination criteria based on the variance of the function value. If this criteria is triggered, the status of the optimization is set to 'tolvariance'. This criteria is suggested by Nelder and Mead.

'-tolabsolutevariance' The absolute tolerance on the variance of the function values of the simplex.

'-tolrelativevariance' The relative tolerance on the variance of the function values of the simplex.

'-kelleystagnationflag' Set to TRUE to enable the termination criteria using Kelley's stagnation detection, based on sufficient decrease condition. If this criteria is triggered, the status of the optimization is set to 'kelleystagnation'.

'-kelleynormalizationflag' Set to FALSE to disable the normalization of the alpha coefficient in Kelley's stagnation detection, i.e. use the value of the `kelleystagnationalpha0` element as is. Default value is TRUE, i.e. the simplex gradient of the initial simplex is takeoptionn into account in the stagnation detection.

'-kelleystagnationalpha0' The parameter used in Kelley's stagnation detection.

'-restartflag' Set to TRUE to enable the automatic restart of the algorithm.

'-restartdetection' The method to detect if the automatic restart must be performed. The following methods are available:

'oneill' The factorial local optimality test by O'Neill is used. If the test finds a local point which is better than the computed optimum, a restart is performed.

'kelley' The sufficient decrease condition by O'Neill is used. If the test finds that the status of the optimization is 'kelleystagnation', a restart is performed. This status may be generated if the -kelleystagnationflag option is set to TRUE.

'-restartmax' The maximum number of restarts, when automatic restart is enabled via the -restartflag option.

'-restarteps' The absolute epsilon value used to check for optimality in the factorial O'Neill restart detection.

'-restartstep' The absolute step length used to check for optimality in the factorial O'Neill restart detection.

'-restartsimplexmethod' The method to compute the initial simplex after a restart. The following methods are available.

'given' The coordinates associated with the `coords0` element are used to compute the initial simplex, with arbitrary number of vertices. This allow the user to setup the initial simplex by a specific method which is not provided by the current package (for example with a simplex computed from a design of experiments). This allows also to configure the initial simplex so that a specific behaviour of the algorithm is to be reproduced (for example the Mc Kinnon test case). The given matrix is expected to have nbve rows and n columns, where n is the dimension of the problem and nbve is the number of vertices.

'axes' The simplex is computed from the coordinate axes and the length associated with the -simplex0length option.

'spendley' The simplex is computed so that it is regular with the length associated with the -simplex0length option (i.e. all the edges have the same length).

'pfeffer' The simplex is computed from an heuristic, in the neighborhood of the initial guess. This initial simplex depends on the -simplex0deltausual and -simplex0deltazero.

'randbounds' The simplex is computed from the bounds and a random number. This option is available only if bounds are available: if bounds are not available, an error is generated. This method is usually associated with Box's algorithm. The number of vertices in the simplex is taken from the -boxnbpoints option.

'oriented' The simplex is computed so that it is oriented, as suggested by Kelley.

'-scalingsimplex0' The algorithm used to scale the initial simplex into the nonlinear constraints. The following two algorithms are provided:

'tox0' scales the vertices toward the initial guess.

'tocentroid' scales the vertices toward the centroid, as recommended by Box.

If the centroid happens to be unfeasible, because the constraints are not convex, the scaling of the initial simplex toward the centroid may fail. Since the initial guess is always feasible, scaling toward the initial guess cannot fail.

'-boxnbpoints' The number of points in the initial simplex, when the -simplex0method is set to 'randbounds'. The value of this option is also use to update the simplex when a restart is

performed and the -restartsimplexmethod option is set to 'randbounds'. The default value is so that the number of points is twice the number of variables of the problem.

'-boxineqscaling' The scaling coefficient used to scale the trial point for function improvement or into the constraints of Box's algorithm.

'-guinalphamin' The minimum value of alpha when scaling the vertices of the simplex into nonlinear constraints in Box's algorithm.

'-boxreflect' The reflection factor in Box's algorithm.

'-boxtermination' Set to TRUE to enable Box's termination criteria.

'-boxtolf' The absolute tolerance on difference of function values in the simplex, suggested by Box. This tolerance is used if the `-boxtermination` element is set to TRUE.

'-boxnbmatch' tThe number of consecutive match of Box's termination criteria.

'-boxboundsalpha' The parameter used to project the vertices into the bounds in Box's algorithm.

'-mymethod' A user-derined simplex algorithm. See `vignette('neldermead',package='neldermead')` for details.

'-myterminate' A user-defined terminate function. See `vignette('neldermead',package='neldermead')` for details.

'-myterminateflag' Set to TRUE to enable the user-defined terminate function.

Value

An updated neldermead object.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`neldermead.new`

`neldermead.destroy` *Erase a neldermead object.*

Description

`neldermead.destroy` calls `optimbase.destroy` and `optimsimplex.destroy` to erase the content of `this$optbase` and `this$simplex0`.

Usage

```
neldermead.destroy(this = NULL)
```

Arguments

`this`	A neldermead object.

Value

Return an updated neldermead object.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`optimbase.destroy`, `optimsimplex.destroy`

`Get functions` *Get the value for the given key*

Description

Get the value for the given key in a neldermead object.

Usage

```
neldermead.get(this = NULL, key = NULL)
neldermead.cget(this = NULL, key = NULL)
```

Arguments

`this`	A neldermead object.
`key`	The name of the key to quiery. The list of available keys for query with `neldermead.get` is: '-historysimplex', '-simplexopt', '-simplex0', and '-restartnb'. If `key` is different, the query is delegated to `optimbase.get`. The list of available keys for query with `neldermead.cget` is: '-method', '-coords0', '-simplex0method', '-simplex0length', '-simplex0deltausual', '-simplex0deltazero', '-rho', '-chi', '-gamma', '-sigma', '-tolsimplexizeabsolute', '-tolsimplexizerelative', '-tolsimplexizemethod', '-toldeltafv', '-tolssizedeltafvmethod', '-restartmax', '-restarteps', '-restartstep', '-kelleystagnationflag', '-kelleynormalizationflag', '-kelleystagnationalpha0', '-restartflag', '-restartdetection', '-restartsimplexmethod', '-checkcostfunction', '-boxnbpoints', '-boxineqscaling', '-scalingsimplex0', '-guinalphamin', '-boxtermination', '-boxtolf', '-boxnbmatch', '-boxreflect', '-mymethod', '-myterminate', '-myterminateflag', '-tolvarianceflag', '-tolabsolutevariance', '-tolrelativevariance', and '-greedy'. If `key` is different, the query is delegated to `optimbase.cget`.

Value

Return the value of the list element `key`, or an error message if `key` does not exist.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`neldermead.configure`, `optimbase.cget`, `optimbase.get`

`neldermead.new` *Initialize Neldermead Object*

Description

Creates a new neldermead object.

Usage

```
neldermead.new()
```

Value

Return a new neldermead object, i.e. a list with a 'type' attribute set to 'T_NELDERMEAD' and containing the following elements:

optbase An optimization object, created by `optimbase.new()`, i.e. list of 'type' attribute set to 'T_OPTIMIZATION' and containing the following elements:

verbose The verbose option, controlling the amount of messages. Set to 0.

x0 The initial guess. Set to NULL.

fx0 The value of the function for the initial guess. Set to NULL.

xopt The optimum parameter. Set to 0.

fopt The optimum function value. Set to 0.

tolfunabsolute The absolute tolerance on function value. Default is 0.

tolfunrelative The relative tolerance on function value. Default is .Machine$double.eps.

tolfunmethod Logical flag for the tolerance on function value in the termination criteria. This criteria is suitable for functions which minimum is associated with a function value equal to 0. Set to FALSE.

tolxabsolute The absolute tolerance on x. Set to 0.

tolxrelative The relative tolerance on x. Set to .Machine$double.eps.

tolxmethod Possible values: FALSE, TRUE. Set to TRUE.

funevals The number of function evaluations. Set to 0.

maxfunevals The maximum number of function evaluations. Set to 100.

maxiter The maximum number of iterations. Set to 100.

iterations The number of iterations. Set to 0.

fun The cost function. Set to ".

status The status of the optimization. Set to ".

historyfopt The vector to store the history for fopt. The values of the cost function will be stored at each iteration in a new element, so the length of `historyfopt` at the end of the optimization should be the number of iterations. Set to NULL.

historyxopt The list to store the history for xopt. The vectors of estimates will be stored on separated levels of the list, so the length of `historyfopt` at the end of the optimization should be the number of iterations. Set to NULL.

verbosetermination The verbose option for termination criteria. Set to 0.

outputcommand The command called back for output. Set to ".

outputcommandarg The outputcommand argument is initialized as a string. If the user configure this element, it is expected that a matrix of values or a list is passed so that the argument is appended to the name of the function. Set to ".

numberofvariables The number of variables to optimize. Set to 0.

storehistory The flag which enables/disables the storing of the history. Set to FALSE.

costfargument The costf argument is initialized as a string. If the user configure this element, it is expected that a matrix of values or a list is passed so that the argument is appended to the name of the function. Set to ".

boundsmin Minimum bounds for the parameters. Set to NULL.

boundsmax Maximum bounds for the parameters. Set to NULL.

nbineqconst The number of nonlinear inequality constraints. Default is 0.

logfile The name of the log file. Set to ".

logfilehandle The handle for the log file. Set to 0.

logstartup Set to TRUE when the logging is started up. Set to FALSE.

withderivatives Set to TRUE when the method uses derivatives. Set to FALSE.

method The name of the algorithm to use. Set to 'variable'.

simplex0 A simplex object created by `optimsimplex.new()$newobj`, i.e. list of type attribute set to 'T_SIMPLEX' and containing the following elements:

verbose The verbose option, controlling the amount of messages. Set to 0.

x The coordinates of the vertices, with size nbve x n. Set to NULL.

n The dimension of the space. Set to 0.

fv The function values, with size nbve x 1. Set to NULL.

nbve The number of vertices. Set to 0.

simplex0method The method to use to compute the initial simplex. Set to 'axes'.

simplex0length The length to use when the initial simplex is computed with the 'axes' or 'spendley' methods. Set to 1.

rho The reflection coefficient. This parameter is used when the `method` element is set to 'fixed' or 'variable'. Set to 1.

chi The expansion coefficient. This parameter is used when the `method` element is set to 'variable'. Set to 2.

gamma The contraction coefficient. This parameter is used when the `method` element is set to 'variable'. Set to 0.5.

sigma The shrinkage coefficient. This parameter is used when the `method` element is set to 'fixed' or 'variable'. Set to 0.5.

tolfstdeviation The tolerance for the standard deviation. Set to 0.

tolfstdeviationmethod Set to FALSE.

tolsimplexizeabsolute The absolute tolerance on the simplex size. Set to 0.

tolsimplexizerelative The relative tolerance on the simplex size. Set to .Machine$double.eps.

tolsimplexizemethod Logical flag to enable/disable the tolerance on the simplex size. When this criteria is enabled, the values of the `tolsimplexizeabsolute` and `tolsimplexizerelative` elements are used in the termination criteria. The method to compute the size is the 'sigmaplus' method. Set to FALSE.

simplexsize0 Initial size of the simplex, for the tolerance on the simplex size. Set to 0.

toldeltafv The absolute tolerance on the difference between the highest and the lowest function values. Set to .Machine$double.eps.

tolssizedeltafvmethod Logical flag to enable/disable the termination criteria based on the size of the simplex and the difference of function value in the simplex. If this criteria is triggered, the status of the optimization is set to 'tolsizedeltafv'. This termination criteria uses the values of the `tolsimplexizeabsolute` and `toldeltafv` elements. This criteria is identical to Scilab's `fminsearch`. Set to FALSE.

historysimplex The list to store the history for simplex. The simplex will be stored on a new level of the list at each iteration, so the length of `historyfopt` at the end of the optimization should be the number of iterations. Set to NULL.

coords0 The coordinates of the vertices of the initial simplex. If the `simplex0method` element is set to 'given', these coordinates are used to compute the initial simplex. This matrix is expected to have shape nbve x n where nbve is the number of vertices and n is the number of variables. Set to NULL.

simplex0deltausual The relative delta for non-zero parameters in 'pfeffer' method. Set to 0.05.

simplex0deltazero The absolute delta for non-zero parameters in 'pfeffer' method. Set to 0.0075.

simplexopt The optimum simplex, after one optimization process. Set to NULL.

restartsimplexmethod The method to compute the initial simplex after a restart. Set to 'oriented'.

restartmax The maximum number of restarts, when automatic restart is enabled via the `restartflag` element. Set to 3.

restarteps The absolute epsilon value used to check for optimality in the factorial O'Neill restart detection. Set to .Machine$double.eps.

restartstep The absolute step length used to check for optimality in the factorial O'Neill restart detection. Set to 1.

kelleystagnationflag Logical flag to enable/disable the termination criteria using Kelley's stagnation detection, based on sufficient decrease condition. If this criteria is triggered, the status of the optimization is set to 'kelleystagnation'. Set to FALSE.,

kelleynormalizationflag Logical flag to enable/disable the normalization of the alpha coefficient in Kelley's stagnation detection, i.e. use the value of the `kelleystagnationalpha0` element as is. Set to TRUE, i.e. the simplex gradient of the initial simplex is taken into account in the stagnation detection.

kelleystagnationalpha0 The parameter used in Kelley's stagnation detection. Set to 1.e-4.

kelleyalpha The current value of Kelley's alpha, after normalization, if required. Set to 1.e-4.

restartnb Number of restarts performed. Set to 0.

restartflag Logical flag to enable/disable the automatic restart of the algorithm. Set to FALSE.

restartdetection The method to detect if the automatic restart must be performed. Set to 'oneill'.

startupflag Set to TRUE when the startup has been performed. Set to FALSE.

boxnbpoints The number of points in the initial simplex, when the `simplex0method` is set to 'randbounds'. The value of this element is also use to update the simplex when a restart is performed and the `restartsimplexmethod` element is set to 'randbounds'. The default value is so that the number of points is twice the number of variables of the problem. Set to '2n'.

boxnbpointseff The effective number of points required in the simplex for Box's algorithm. Set to 0.

boxineqscaling The scaling coefficient used to scale the trial point for function improvement or into the constraints of Box's algorithm. Set to 0.

checkcostfunction Logical flag to enable/disable the checking of the connection of the cost function. Set to TRUE.

scalingsimplex0 The algorithm used to scale the initial simplex into the nonlinear constraints. The following two algorithms are provided:

'tox0' scales the vertices toward the initial guess.

'tocentroid' scales the vertices toward the centroid, as recommended by Box.

If the centroid happens to be unfeasible, because the constraints are not convex, the scaling of the initial simplex toward the centroid may fail. Since the initial guess is always feasible, scaling toward the initial guess cannot fail. Set to 'tox0'.

guinalphamin The minimum value of alpha when scaling the vertices of the simplex into nonlinear constraints in Box's algorithm. Set to 1.e-5.

boxboundsalpha The parameter used to project the vertices into the bounds in Box's algorithm. Set to 1.e-6.

boxtermination Logical flag to enable/disable Box's termination criteria. Set to FALSE.

boxtolf The absolute tolerance on difference of function values in the simplex, suggested by Box. This tolerance is used if the `boxtermination` element is set to TRUE. Set to 1.e-5.

boxnbmatch The number of consecutive match of Box's termination criteria. Set to 5.

boxkount Current number of consecutive match. Set to 0.

boxreflect The reflection factor in Box's algorithm. Set to 1.3.

tolvarianceflag Logical flag to enable/disable the termination criteria based on the variance of the function value. If this criteria is triggered, the status of the optimization is set to 'tolvariance'. This criteria is suggested by Nelder and Mead. Set to FALSE.

tolabsolutevariance The absolute tolerance on the variance of the function values of the simplex. Set to 0.

tolrelativevariance The relative tolerance on the variance of the function values of the simplex. Set to .Machine$double.eps.

variancesimplex0 Relative tolerance on variance. Set to .Machine$double.eps.

mymethod A user-derined simplex algorithm. Set to NULL.

myterminate A user-defined terminate function. Set to NULL.

myterminateflag Logical flag to enable/disable the user-defined terminate function. Set to FALSE.

greedy Logical flag to enable/disable greedy Nelder-Mead. Set to FALSE.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`optimbase.new`, `optimsimplex.new`

`neldermead.restart` *Restart neldermead search.*

Description

Update the simplex with `neldermead.updatesimp` and restart the search with `neldermead.search`.

Usage

```
neldermead.restart(this = NULL)
```

Arguments

`this`	A neldermead object.

Value

Returns an updated neldermead object.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`neldermead.updatesimp`, `neldermead.search`,

`neldermead.search` *Starts the optimization*

Description

Performs the optimization associated with the method associated with the `method` element of the neldermead object and find the optimum. If the `restartflag` element is enabled, automatic restarts are performed, based on the `restartdetection` element.

Usage

```
neldermead.search(this = NULL)
```

Arguments

`this`	A neldermead object.

Value

Return an updated neldermead object.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`fminsearch`, `neldermead.new`, `neldermead.configure`,

`Secondary search functions`
Secondary functions for neldermead.search

Description

Utility functions for `neldermead.serch` and dependent functions.

Usage

```
neldermead.startup(this = NULL)
neldermead.log(this = NULL, msg = NULL)
neldermead.scaletox0(this = NULL, simplex0 = NULL)
neldermead.scaletocenter(this = NULL, simplex0 = NULL, x0 = NULL)
neldermead.termstartup(this = NULL)
neldermead.outputcmd(this = NULL, state = NULL, simplex = NULL, step = NULL)
neldermead.autorestart(this = NULL)
neldermead.istorestart(this = NULL)
neldermead.isroneill(this = NULL)
neldermead.isrkelley(this = this)
neldermead.updatesimp(this = NULL)
scaleinconstraints(this = NULL, x = NULL, xref = NULL)
neldermead.costf(x = NULL, this = NULL)
```

Arguments

`this`	A neldermead object.
`msg`	A character string.
`simplex0`	The initial simplex object.
`x0`	A column matrix of initial parameters.
`state`	The state of the algorithm, either 'init', 'done' or 'iter'.
`simplex`	The current simplex object.
`step`	The type of step performed during the iteration: 'init', 'done', 'reflection', 'expansion', 'insidecontraction', 'outsidecontraction', 'reflectionnext' or 'shrink'.
`x`	The point estimate to scale.
`xref`	The reference point estimate.

Details

`neldermead.startup` Startup the algorithm. Compute the initial simplex, depending on the content of the `simplex0method` element of the neldermead object ('given', 'axes', 'spendley', 'pfeffer' or 'randbounds').

`neldermead.log` Print a message to the log file using `optimbase.log`.

`neldermead.scaletox0` Scale the simplex into the nonlinear inequality constraints, if any. Scale toward x0, which is feasible.

`neldermead.scaletocenter` Scale the simplex into the nonlinear inequality constraints, if any. Scale to the centroid of the points which satisfy the constraints. This is Box's method for scaling. It is unsure, since the centroid of the points which satisfy the constraints may not be feasible.

`neldermead.termstartup` Initialize Kelley's stagnation detection system when normalization is required, by computing kelleyalpha. If the simplex gradient is zero, then use alpha0 as alpha.

`neldermead.outputcmd` Call the array of user-defined output functions

`neldermead.autorestart` Perform an optimization with automatic restart. The loop processes for i = 1 to `restartmax` + 1. This is because a RE-start is performed after one simulation has been performed, hence the 'RE'.

`neldermead.istorestart` Determine if the optimization is to restart using `neldermead.isroneill` or `neldermead.isrkelley` depending on the content of the `restartdetection` element.

`neldermead.isroneill` Determine if the optimization is to restart. Use O'Neill method as a criteria for restart. It is an axis-by-axis search for optimality.

`neldermead.isrkelley` Determine if the optimization is to restart. Use `kelleystagnation` as a criteria for restart.

`neldermead.updatesimp` Update the initial simplex simplex0 for a restart.

`scaleinconstraints` Given a point reference to scale and a reference point which satisfies the constraints, scale the point towards the reference point estimate until it satisfies all the constraints.

`neldermead.costf` Call the cost function and return the value. This function is given to the simplex function class as a callback. Input/Output arguments are swapped w.r.t. `optimbase.function`, so that it matches the requirements of simplex methods.

Value

`neldermead.startup` Return an updated neldermead object `this`.

`neldermead.log` Return the neldermead object `this`.

`neldermead.scaletox0` Return an updated simplex.

`neldermead.scaletocenter` Return an updated simplex.

`neldermead.termstartup` Return an updated neldermead object `this`.

`neldermead.outputcmd` Do not return any data, but execute the output function(s).

`neldermead.autorestart` Return an updated neldermead object `this`.

`neldermead.istorestart` Return a list with the following elements:

this The input neldermead object.

istorestart Set to TRUE if the optimization is to restart, to FALSE otherwise.

`neldermead.isroneill` Return a list with the following elements:

this The input neldermead object.

istorestart Set to TRUE if the optimization is to restart, to FALSE otherwise.

`neldermead.isrkelley` Return a list with the following elements:

this The input neldermead object.

istorestart Set to TRUE if the optimization is to restart, to FALSE otherwise.

`neldermead.updatesimp` Return an updated neldermead object `this`.

`scaleinconstraints` Return a list with the following elements:

this The updated neldermead object.

isscaled TRUE if the procedure has succeeded before `boxnbnlloops`, FALSE if it has failed.

p The scaled parameters.

`neldermead.costf` Return a list with the following elements:

f The value of the cost function.

this The updated neldermead object.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

`optimget` *Queries an optimization option list*

Description

This function allows to make queries on an existing optimization option list. This list must have been created and updated by the `optimset` function. The `optimget` allows to retrieve the value associated with a given key.

Usage

```
optimget(options = NULL, key = NULL, value = NULL)
```

Arguments

`options`	A list created or modifies by `optimset`.
`key`	A single character string, which should be the name of the field in `options` to query (case insensitive).
`value`	A default value.

Details

`key` is matched against the field names of `options` using `grep` and a case-insensitive regular expression. If `key` is not found in `options`, the function returns NULL. If several matches are found, `optimget` is stopped.

Value

Return `options$key` if `key` is found in `options`. Return `value`, otherwise.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`optimset`

Examples

```
opt <- optimset(method='fminsearch')
optimget(opt,'Display')
optimget(opt,'abc','!@')
```

`optimset.method` *Default set of optimization options*

Description

This function returns a default set of optimization options for defined 'methods'; `optimset.method` is called by `optimset` when a `method` was provided as input. Currently, the only valid `method` is 'fminsearch'.

Usage

```
optimset.method(method = NULL)
```

Arguments

`method`	A character string.

Value

Returns a list with the following fields: Display, FunValCheck, MaxFunEvals, MaxIter, OutputFcn, PlotFcns, TolFun, and TolX.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`optimset`

Examples

```
optimset.method('fminsearch')
## Not run: optimset.method('abc')
```

`optimset` *Configures and returns an optimization data structure.*

Description

This function creates or updates a list which can be used to modify the behaviour of optimization methods. The goal of this function is to manage the `options` list with a set of fields (for example, 'MaxFunEvals', 'MaxIter', etc...). The user can create a new list with empty fields or create a new structure with default fields which correspond to a particular algorithm. The user can also configure each field and set it to a particular value. Finally, the user passes the list to an optimization function so that the algorithm uses the options configured by the user.

Usage

```
optimset(method = NULL,
         Display = NULL,
         FunValCheck = NULL,
         MaxFunEvals = NULL,
         MaxIter = NULL,
         OutputFcn = NULL,
         PlotFcns = NULL,
         TolFun = NULL,
         TolX = NULL)
```

Arguments

`method`	If provided, the `method` argument overrides all the others and `optimset.method` is called. If the content of `method` is recognized, a defuault set of options are returned. The only current recognized character string is 'fminsearch'.
`Display`	The verbose level. The default value is 'notify'. The following is a list of available verbose levels. **'off'** The algorithm displays no message at all. **'notify'** The algorithm displays message if the termination criteria is not reached at the end of the optimization. This may happen if the maximum number or iterations of the maximum number of function evaluations is reached and warns the user of a convergence problem. **'final'** The algorithm displays a message at the end of the optimization, showing the number of iterations, the number of function evaluations and the status of the optimization. This option includes the messages generated by the 'notify' option i.e. warns in case of a convergence problem. **'iter'** The algorithm displays a one-line message at each iteration. This option includes the messages generated by the 'notify' option i.e. warns in case of a convergence problem. It also includes the message generated by the 'final' option.
`FunValCheck`	A logical flag to enable the checking of function values.
`MaxFunEvals`	The maximum number of evaluations of the cost function.
`MaxIter`	The maximum number of iterations.
`OutputFcn`	A function which is called at each iteration to print out intermediate state of the optimization algorithm (for example into a log file).
`PlotFcns`	A function which is called at each iteration to plot the intermediate state of the optimization algorithm (for example into a 2D graphic).
`TolFun`	The absolute tolerance on function value.
`TolX`	The absolute tolerance on the variable x.

Details

Most optimization algorithms require many algorithmic parameters such as the number of iterations or the number of function evaluations. If these parameters are given to the optimization

function as input parameters, this forces both the user and the developper to manage many input parameters. The goal of the `optimset` function is to simplify the management of input arguments, by gathering all the parameters into a single list.

While the current implementation of the `optimset` function only supports the `fminsearch` function, it is designed to be extended to as many optimization function as required. Because all optimization algorithms do not require the same parameters, the data structure aims at remaining flexible. But, most of the time, most parameters are the same from algorithm to algorithm, for example, the tolerance parameters which drive the termination criteria are often the same, even if the termination criteria itself is not the same.

Output and plot functions The 'OutputFcn' and 'PlotFcns' options accept as argument a function (or a list of functions). In the client optimization algorithm, this output or plot function is called back once per iteration. It can be used by the user to display a message in the console, write into a file, etc... The output or plot function is expected to have the following definition:

myfun <- function(x, optimValues, state)

where the input parameters are:

x The current point estimate.

optimValues A list which contains the following fields:

funccount The number of function evaluations.

fval The best function value.

iteration The current iteration number.

procedure The type of step performed. This string depends on the specific algorithm (see `fminsearch` for details).

state the state of the algorithm. The following states are available:

'init' when the algorithm is initializing,

'iter' when the algorithm is performing iterations,

'done' when the algorithm is terminated.

Value

Return a list with the following fields: Display, FunValCheck, MaxFunEvals, MaxIter, OutputFcn, PlotFcns, TolFun, and TolX.

Author(s)

Author of Scilab neldermead module: Michael Baudin (INRIA - Digiteo)

Author of R adaptation: Sebastien Bihorel (<sb.pmlab@gmail.com>)

See Also

`optimset.method`,`fminsearch`

Examples

```
optimset()
optimset(Display='iter')
```

7 CeCILL FREE SOFTWARE LICENSE AGREEMENT

Notice

This Agreement is a Free Software license agreement that is the result of discussions between its authors in order to ensure compliance with the two main principles guiding its drafting:

* firstly, compliance with the principles governing the distribution of Free Software: access to source code, broad rights granted to users,
* secondly, the election of a governing law, French law, with which it is conformant, both as regards the law of torts and intellectual property law, and the protection that it offers to both authors and holders of the economic rights over software.

The authors of the CeCILL (for Ce[a] C[nrs] I[nria] L[ogiciel] L[ibre]) license are:

Commissariat a l'Energie Atomique - CEA, a public scientific, technical and industrial research establishment, having its principal place of business at 25 rue Leblanc, immeuble Le Ponant D, 75015 Paris, France.

Centre National de la Recherche Scientifique - CNRS, a public scientific and technological establishment, having its principal place of business at 3 rue Michel-Ange, 75794 Paris cedex 16, France.

Institut National de Recherche en Informatique et en Automatique - INRIA, a public scientific and technological establishment, having its principal place of business at Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay cedex, France.

Preamble

The purpose of this Free Software license agreement is to grant users the right to modify and redistribute the software governed by this license within the framework of an open source distribution model.

The exercising of these rights is conditional upon certain obligations for users so as to preserve this status for all subsequent redistributions.

In consideration of access to the source code and the rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors only have limited liability.

In this respect, the risks associated with loading, using, modifying and/or developing or reproducing the software by the user are brought to

the user's attention, given its Free Software status, which may make it complicated to use, with the result that its use is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the suitability of the software as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions of security. This Agreement may be freely reproduced and published, provided it is not altered, and that no provisions are either added or removed herefrom.

This Agreement may apply to any or all software for which the holder of the economic rights decides to submit the use thereof to its provisions.

Article 1 - DEFINITIONS

For the purpose of this Agreement, when the following expressions commence with a capital letter, they shall have the following meaning:

Agreement: means this license agreement, and its possible subsequent versions and annexes.

Software: means the software in its Object Code and/or Source Code form and, where applicable, its documentation, "as is" when the Licensee accepts the Agreement.

Initial Software: means the Software in its Source Code and possibly its Object Code form and, where applicable, its documentation, "as is" when it is first distributed under the terms and conditions of the Agreement.

Modified Software: means the Software modified by at least one Contribution.

Source Code: means all the Software's instructions and program lines to which access is required so as to modify the Software.

Object Code: means the binary files originating from the compilation of the Source Code.

Holder: means the holder(s) of the economic rights over the Initial Software.

Licensee: means the Software user(s) having accepted the Agreement.

Contributor: means a Licensee having made at least one Contribution.

Licensor: means the Holder, or any other individual or legal entity, who

distributes the Software under the Agreement.

Contribution: means any or all modifications, corrections, translations, adaptations and/or new functions integrated into the Software by any or all Contributors, as well as any or all Internal Modules.

Module: means a set of sources files including their documentation that enables supplementary functions or services in addition to those offered by the Software.

External Module: means any or all Modules, not derived from the Software, so that this Module and the Software run in separate address spaces, with one calling the other when they are run.

Internal Module: means any or all Module, connected to the Software so that they both execute in the same address space.

GNU GPL: means the GNU General Public License version 2 or any subsequent version, as published by the Free Software Foundation Inc.

Parties: mean both the Licensee and the Licensor.

These expressions may be used both in singular and plural form.

Article 2 - PURPOSE

The purpose of the Agreement is the grant by the Licensor to the Licensee of a non-exclusive, transferable and worldwide license for the Software as set forth in Article 5 hereinafter for the whole term of the protection granted by the rights over said Software.

Article 3 - ACCEPTANCE

3.1 The Licensee shall be deemed as having accepted the terms and conditions of this Agreement upon the occurrence of the first of the following events:

* (i) loading the Software by any or all means, notably, by downloading from a remote server, or by loading from a physical medium;
* (ii) the first time the Licensee exercises any of the rights granted hereunder.

3.2 One copy of the Agreement, containing a notice relating to the characteristics of the Software, to the limited warranty, and to the fact that its use is restricted to experienced users has been provided

to the Licensee prior to its acceptance as set forth in Article 3.1 hereinabove, and the Licensee hereby acknowledges that it has read and understood it.

Article 4 - EFFECTIVE DATE AND TERM

4.1 EFFECTIVE DATE

The Agreement shall become effective on the date when it is accepted by the Licensee as set forth in Article 3.1.

4.2 TERM

The Agreement shall remain in force for the entire legal term of protection of the economic rights over the Software.

Article 5 - SCOPE OF RIGHTS GRANTED

The Licensor hereby grants to the Licensee, who accepts, the following rights over the Software for any or all use, and for the term of the Agreement, on the basis of the terms and conditions set forth hereinafter.

Besides, if the Licensor owns or comes to own one or more patents protecting all or part of the functions of the Software or of its components, the Licensor undertakes not to enforce the rights granted by these patents against successive Licensees using, exploiting or modifying the Software. If these patents are transferred, the Licensor undertakes to have the transferees subscribe to the obligations set forth in this paragraph.

5.1 RIGHT OF USE

The Licensee is authorized to use the Software, without any limitation as to its fields of application, with it being hereinafter specified that this comprises:

1. permanent or temporary reproduction of all or part of the Software by any or all means and in any or all form.

2. loading, displaying, running, or storing the Software on any or all medium.

3. entitlement to observe, study or test its operation so as to

determine the ideas and principles behind any or all constituent elements of said Software. This shall apply when the Licensee carries out any or all loading, displaying, running, transmission or storage operation as regards the Software, that it is entitled to carry out hereunder.

5.2 ENTITLEMENT TO MAKE CONTRIBUTIONS

The right to make Contributions includes the right to translate, adapt, arrange, or make any or all modifications to the Software, and the right to reproduce the resulting software.

The Licensee is authorized to make any or all Contributions to the Software provided that it includes an explicit notice that it is the author of said Contribution and indicates the date of the creation thereof.

5.3 RIGHT OF DISTRIBUTION

In particular, the right of distribution includes the right to publish, transmit and communicate the Software to the general public on any or all medium, and by any or all means, and the right to market, either in consideration of a fee, or free of charge, one or more copies of the Software by any means.

The Licensee is further authorized to distribute copies of the modified or unmodified Software to third parties according to the terms and conditions set forth hereinafter.

5.3.1 DISTRIBUTION OF SOFTWARE WITHOUT MODIFICATION

The Licensee is authorized to distribute true copies of the Software in Source Code or Object Code form, provided that said distribution complies with all the provisions of the Agreement and is accompanied by:

1. a copy of the Agreement,

2. a notice relating to the limitation of both the Licensor's warranty and liability as set forth in Articles 8 and 9,

and that, in the event that only the Object Code of the Software is redistributed, the Licensee allows future Licensees unhindered access to the full Source Code of the Software by indicating how to access it, it being understood that the additional cost of acquiring the Source Code shall not exceed the cost of transferring the data.

5.3.2 DISTRIBUTION OF MODIFIED SOFTWARE

When the Licensee makes a Contribution to the Software, the terms and conditions for the distribution of the resulting Modified Software become subject to all the provisions of this Agreement.

The Licensee is authorized to distribute the Modified Software, in source code or object code form, provided that said distribution complies with all the provisions of the Agreement and is accompanied by:

1. a copy of the Agreement,

2. a notice relating to the limitation of both the Licensor's warranty and liability as set forth in Articles 8 and 9,

and that, in the event that only the object code of the Modified Software is redistributed, the Licensee allows future Licensees unhindered access to the full source code of the Modified Software by indicating how to access it, it being understood that the additional cost of acquiring the source code shall not exceed the cost of transferring the data.

5.3.3 DISTRIBUTION OF EXTERNAL MODULES

When the Licensee has developed an External Module, the terms and conditions of this Agreement do not apply to said External Module, that may be distributed under a separate license agreement.

5.3.4 COMPATIBILITY WITH THE GNU GPL

The Licensee can include a code that is subject to the provisions of one of the versions of the GNU GPL in the Modified or unmodified Software, and distribute that entire code under the terms of the same version of the GNU GPL.

The Licensee can include the Modified or unmodified Software in a code that is subject to the provisions of one of the versions of the GNU GPL, and distribute that entire code under the terms of the same version of the GNU GPL.

Article 6 - INTELLECTUAL PROPERTY

6.1 OVER THE INITIAL SOFTWARE

The Holder owns the economic rights over the Initial Software. Any or all use of the Initial Software is subject to compliance with the terms and conditions under which the Holder has elected to distribute its work and no one shall be entitled to modify the terms and conditions for the distribution of said Initial Software.

The Holder undertakes that the Initial Software will remain ruled at least by this Agreement, for the duration set forth in Article 4.2.

6.2 OVER THE CONTRIBUTIONS

The Licensee who develops a Contribution is the owner of the intellectual property rights over this Contribution as defined by applicable law.

6.3 OVER THE EXTERNAL MODULES

The Licensee who develops an External Module is the owner of the intellectual property rights over this External Module as defined by applicable law and is free to choose the type of agreement that shall govern its distribution.

6.4 JOINT PROVISIONS

The Licensee expressly undertakes:

1. not to remove, or modify, in any manner, the intellectual property notices attached to the Software;

2. to reproduce said notices, in an identical manner, in the copies of the Software modified or not.

The Licensee undertakes not to directly or indirectly infringe the intellectual property rights of the Holder and/or Contributors on the Software and to take, where applicable, vis-a-vis its staff, any and all measures required to ensure respect of said intellectual property rights of the Holder and/or Contributors.

Article 7 - RELATED SERVICES

7.1 Under no circumstances shall the Agreement oblige the Licensor to provide technical assistance or maintenance services for the Software.

However, the Licensor is entitled to offer this type of services. The terms and conditions of such technical assistance, and/or such maintenance, shall be set forth in a separate instrument. Only the Licensor offering said maintenance and/or technical assistance services shall incur liability therefor.

7.2 Similarly, any Licensor is entitled to offer to its licensees, under its sole responsibility, a warranty, that shall only be binding upon itself, for the redistribution of the Software and/or the Modified Software, under terms and conditions that it is free to decide. Said warranty, and the financial terms and conditions of its application, shall be subject of a separate instrument executed between the Licensor and the Licensee.

Article 8 - LIABILITY

8.1 Subject to the provisions of Article 8.2, the Licensee shall be entitled to claim compensation for any direct loss it may have suffered from the Software as a result of a fault on the part of the relevant Licensor, subject to providing evidence thereof.

8.2 The Licensor's liability is limited to the commitments made under this Agreement and shall not be incurred as a result of in particular: (i) loss due the Licensee's total or partial failure to fulfill its obligations, (ii) direct or consequential loss that is suffered by the Licensee due to the use or performance of the Software, and (iii) more generally, any consequential loss. In particular the Parties expressly agree that any or all pecuniary or business loss (i.e. loss of data, loss of profits, operating loss, loss of customers or orders, opportunity cost, any disturbance to business activities) or any or all legal proceedings instituted against the Licensee by a third party, shall constitute consequential loss and shall not provide entitlement to any or all compensation from the Licensor.

Article 9 - WARRANTY

9.1 The Licensee acknowledges that the scientific and technical state-of-the-art when the Software was distributed did not enable all possible uses to be tested and verified, nor for the presence of possible defects to be detected. In this respect, the Licensee's attention has been drawn to the risks associated with loading, using, modifying and/or developing and reproducing the Software which are reserved for experienced users.

The Licensee shall be responsible for verifying, by any or all means, the suitability of the product for its requirements, its good working

order, and for ensuring that it shall not cause damage to either persons or properties.

9.2 The Licensor hereby represents, in good faith, that it is entitled to grant all the rights over the Software (including in particular the rights set forth in Article 5).

9.3 The Licensee acknowledges that the Software is supplied "as is" by the Licensor without any other express or tacit warranty, other than that provided for in Article 9.2 and, in particular, without any warranty as to its commercial value, its secured, safe, innovative or relevant nature.

Specifically, the Licensor does not warrant that the Software is free from any error, that it will operate without interruption, that it will be compatible with the Licensee's own equipment and software configuration, nor that it will meet the Licensee's requirements.

9.4 The Licensor does not either expressly or tacitly warrant that the Software does not infringe any third party intellectual property right relating to a patent, software or any other property right. Therefore, the Licensor disclaims any and all liability towards the Licensee arising out of any or all proceedings for infringement that may be instituted in respect of the use, modification and redistribution of the Software. Nevertheless, should such proceedings be instituted against the Licensee, the Licensor shall provide it with technical and legal assistance for its defense. Such technical and legal assistance shall be decided on a case-by-case basis between the relevant Licensor and the Licensee pursuant to a memorandum of understanding. The Licensor disclaims any and all liability as regards the Licensee's use of the name of the Software. No warranty is given as regards the existence of prior rights over the name of the Software or as regards the existence of a trademark.

Article 10 - TERMINATION

10.1 In the event of a breach by the Licensee of its obligations hereunder, the Licensor may automatically terminate this Agreement thirty (30) days after notice has been sent to the Licensee and has remained ineffective.

10.2 A Licensee whose Agreement is terminated shall no longer be authorized to use, modify or distribute the Software. However, any licenses that it may have granted prior to termination of the Agreement shall remain valid subject to their having been granted in compliance with the terms and conditions hereof.

Article 11 - MISCELLANEOUS

11.1 EXCUSABLE EVENTS

Neither Party shall be liable for any or all delay, or failure to perform the Agreement, that may be attributable to an event of force majeure, an act of God or an outside cause, such as defective functioning or interruptions of the electricity or telecommunications networks, network paralysis following a virus attack, intervention by government authorities, natural disasters, water damage, earthquakes, fire, explosions, strikes and labor unrest, war, etc.

11.2 Any failure by either Party, on one or more occasions, to invoke one or more of the provisions hereof, shall under no circumstances be interpreted as being a waiver by the interested Party of its right to invoke said provision(s) subsequently.

11.3 The Agreement cancels and replaces any or all previous agreements, whether written or oral, between the Parties and having the same purpose, and constitutes the entirety of the agreement between said Parties concerning said purpose. No supplement or modification to the terms and conditions hereof shall be effective as between the Parties unless it is made in writing and signed by their duly authorized representatives.

11.4 In the event that one or more of the provisions hereof were to conflict with a current or future applicable act or legislative text, said act or legislative text shall prevail, and the Parties shall make the necessary amendments so as to comply with said act or legislative text. All other provisions shall remain effective. Similarly, invalidity of a provision of the Agreement, for any reason whatsoever, shall not cause the Agreement as a whole to be invalid.

11.5 LANGUAGE

The Agreement is drafted in both French and English and both versions are deemed authentic.

Article 12 - NEW VERSIONS OF THE AGREEMENT

12.1 Any person is authorized to duplicate and distribute copies of this Agreement.

12.2 So as to ensure coherence, the wording of this Agreement is

protected and may only be modified by the authors of the License, who reserve the right to periodically publish updates or new versions of the Agreement, each with a separate number. These subsequent versions may address new issues encountered by Free Software.

12.3 Any Software distributed under a given version of the Agreement may only be subsequently distributed under the same version of the Agreement or a subsequent version, subject to the provisions of Article 5.3.4.

Article 13 - GOVERNING LAW AND JURISDICTION

13.1 The Agreement is governed by French law. The Parties agree to endeavor to seek an amicable solution to any disagreements or disputes that may arise during the performance of the Agreement.

13.2 Failing an amicable solution within two (2) months as from their occurrence, and unless emergency proceedings are necessary, the disagreements or disputes shall be referred to the Paris Courts having jurisdiction, by the more diligent Party.

Version 2.0 dated 2006-09-05.