The optimsimplex Package

Sébastien Bihorel

May 11, 2010

optimsimplex 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 optimsimplex module.

1 Overview

1.1 Description

The goal of this package is to provide a building block for optimization algorithms based on a simplex. The **optimsimplex** package may be used in the following optimization methods:

- the simplex method Spendley *et al.*,
- the method of Nelder and Mead,
- the Box's algorithm for constrained optimization,
- the multi-dimensional search by Torczon,
- etc ...

This set of commands allows to manage a simplex made of $k \geq n+1$ points in a n-dimensional space. This component is the building block for a class of direct search optimization methods such as the Nelder-Mead algorithm or Torczon's Multi-Dimensionnal Search.

A simplex is designed as a collection of $k \geq n+1$ vertices. Each vertex is made of a point and a function value at that point.

The simplex can be created with various shapes. It can be configured and quieried at will. The simplex can also be reflected or shrinked. The simplex gradient can be computed with a order 1 forward formula and with a order 2 centered formula.

The `optimsimplex.new` function allows to create a simplex. If vertices coordinates are given, there are registered in the simplex. If a function is provided, it is evaluated at each vertex. Several functions allow to create a simplex with special shapes and methods, including axes-by-axes (`optimsimplex.axes`), regular (`optimsimplex.spendley`), randomized bounds simplex with arbitrary *nbve* vertices (`optimsimplex.randbounds`) and an heuristical small variation around a given point (`optimsimplex.pfeffer`).

In the functions provided in this package, simplices and vertices are, depending on the functions either input or output arguments. The following general principle have been used to manage the storing of the coordinates of the points.

- The vertices are stored row by row, while the coordinates are stored column by column. This implies the following rules.
- The coordinates of a vertex are stored in a row vector, i.e. a 1 x n matrix where n is the dimension of the space.
- The function values are stored in a column vector, i.e. a *nbve* x 1 matrix where *nbve* is the number of vertices.

1.2 Computation of function value at the given vertices

Most functions in the **optimsimplex** package accept a `fun` argument, which corresponds to the function to be evaluated at the given vertices. The function is expected to have the following input and output arguments:

```
myfunction <- function(x, this){
  ...
  return(list(f=f,this=this))
}
```

where `x` is a row vector, `f` is the function value, and `this` an optional user-defined data passed to the function. If data is provided, it is passed to the callback function both as an input and output argument. `data` may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

2 Examples

2.1 Creating a simplex given vertex coordinates

In the following example, one creates a simplex with known vertices coordinates and queries the new object. The function values at the vertices are unset.

```
> coords <- matrix(c(0, 1, 0, 0, 0, 1), ncol = 2)
> tmp <- optimsimplex.new(coords = coords)
> s1 <- tmp$newobj
> s1

$verbose
[1] 0

$x
     [,1] [,2]
[1,]    0    0
[2,]    1    0
[3,]    0    1

$n
[1] 2
```

```
$fv
     [,1]

$nbve
[1] 3

attr(,"type")
[1] "T_SIMPLEX"

> optimsimplex.getallx(s1)

     [,1] [,2]
[1,]    0    0
[2,]    1    0
[3,]    0    1

> optimsimplex.getn(s1)

[1] 2

> optimsimplex.getnbve(s1)

[1] 3
```

2.2 Creating a simplex with randomized bounds

In the following example, one creates a simplex with in the 2D domain c(-5, 5)^2, with c(-1.2, 1.0) as the first vertex. One uses the randomized bounds method to generate a simplex with 5 vertices. The function takes an additionnal argument `this`, which counts the number of times the function is called. After the creation of the simplex, the value of `this$nb` is 5, which is the expected result because there is one function call by vertex.

```
> rosenbrock <- function(x) {
+     y <- 100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2
+ }
> mycostf <- function(x, this) {
+     y <- rosenbrock(x)
+     this$nb <- this$nb + 1
+     return(list(f = y, this = this))
+ }
> mystuff <- list(nb = 0)
> tmp <- optimsimplex.randbounds(x0 = c(-1.2, 1), fun = mycostf,
+     boundsmin = c(-5, -5), boundsmax = c(5, 5), nbve = 5,
+     data = mystuff)
> tmp$newobj

$verbose
[1] 0
```

```
$x
           [,1]      [,2]
[1,] -1.2000000  1.000000
[2,]  0.8397690  2.357197
[3,] -3.7707211  3.684707
[4,]  0.1965941  2.775247
[5,]  0.3239338 -2.045633

$n
[1] 2

$fv
           [,1]
[1,]    24.2000
[2,]   272.9310
[3,] 11118.4968
[4,]   749.5424
[5,]   462.9504

$nbve
[1] 5

attr(,"type")
[1] "T_SIMPLEX"

> tmp$data

$nb
[1] 5

> cat(sprintf("Function evaluations: %d\n", tmp$data$nb))

Function evaluations: 5
```

3 Initial simplex strategies

In this section, we analyse the various initial simplex which are provided in this component.

It is known that direct search methods based on simplex designs are very sensitive to the initial simplex. This is why the current component provides various ways to create such an initial simplex.

The first historical simplex-based algorithm is the one presented in ”Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation” by W. Spendley, G. R. Hext and F. R. Himsworth. The ”spendley” simplex creates the regular simplex which is presented in the paper [9].

The ”randbounds” simplex is due to M.J. Box in ”A New Method of Constrained Optimization and a Comparison With Other Methods” [7].

Pfeffer’s method is an heuristic which is presented in ”Global Optimization Of Lennard-Jones Atomic Clusters” by E. Fan [4]. It is due to L. Pfeffer at Stanford and it is used in the `fminsearch` function from the **neldermead** package.

4 References

The functions distributed in **optimsimplex** are also based upon the work from Nelder and Mead [5], Kelley [3], Han and Neumann [6], Torczon [8], Burmen et al. [1], and Price and al. [2].

[1] A. Burmen and J. Puhan and T. Tuma. Grid Restrained Nelder-Mead Algorithm. *Computational Optimization and Applications*, 34(3):359–375, July 2006.

[2] C.J. Price and I.D. Coope and D. Byatt. A Convergent Variant of The Nelder-Mead algorithm. *Journal of Optimization Theory and Applications*, 113(1):5–19, April 2002.

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

[4] E. Fan. Global Optimization Of Lennard-Jones Atomic Clusters. Master's thesis, McMaster University, February 2002.

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

[6] Lixing Han and Michael Neumann. Effect of Dimensionality on the Nelder-Mead Simplex Method. *Optimization methods and software*, 21(1):1–16, 2006.

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

[8] V.J. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Rice University, Houston, TX, 1989.

[9] 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 Network of optimsimplex functions

The network of functions provided in **optimsimplex** is illustrated in the network map given in the **neldermead** package.

6 Help on optimsimplex functions

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

Description

The goal of this package is to provide a building block for optimization algorithms based on a simplex. The **optimsimplex** package may be used in the following optimization methods:

- the simplex method of Spendley et al.,
- the method of Nelder and Mead,
- the Box's algorithm for constrained optimization,
- the multi-dimensional search by Torczon,
- etc ...

Features The following is a list of features currently provided:

- Manage various simplex initializations
  - initial simplex given by user,
  - initial simplex computed with a length and along the coordinate axes,
  - initial regular simplex computed with Spendley et al. formula,
  - initial simplex computed by a small perturbation around the initial guess point,
  - initial simplex computed from randomized bounds.
- sort the vertices by increasing function values,
- compute the standard deviation of the function values in the simplex,
- compute the simplex gradient with forward or centered differences,
- shrink the simplex toward the best vertex,
- etc...

Details

Package:	optimsimplex
Type:	Package
Version:	1.0-2
Date:	2010-05-11
License:	CeCILL-2
LazyLoad:	yes

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

Author(s)

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

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

Function evaluations *Computation of Function Value(s)*

Description

These functions compute the value of the function at the vertices points stored in the current simplex object and stored them back into the simplex object. `optimsimplex.computefv` determines how many vertices are stored in the simplex object and delegates the calculation of the function values to `optimsimplex.compsomefv`.

Usage

```
optimsimplex.computefv(this = NULL, fun = NULL, data = NULL)
optimsimplex.compsomefv(this = NULL, fun = NULL, indices = NULL, data = NULL)
```

Arguments

`this` The current simplex object, containing the nbve x n matrix of vertice coordinates (i.e. `x` element), where n is the dimension of the space and nbve the number of vertices.

`fun` The function to compute at vertices. The function is expected to have the following input and output arguments:

```
myfunction <- function(x, this){
 ...
 return(list(f=f,this=this))
 }
```

where x is a row vector and this a user-defined data, i.e. the `data` argument.

`data` A user-defined data passed to the function. If data is provided, it is passed to the callback function both as an input and output argument. `data` may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

`indices` A vector of increasing integers from 1 to nbve.

Value

`optimsimplex.computefv` and `optimsimplex.compsomefv` return a list with the following elements:

this The updated simplex object.

data The updated user-defined data.

Author(s)

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

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

See Also

`optimsimplex.new`

`optimsimplex.destroy` *Erase Simplex Object*

Description

This function erases the coordinates of the vertices (`x`) and the function values (`fv`) in a simplex object

Usage

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

Arguments

`this`	A simplex object.

Value

Return an updated simplex object for which the content of the `x` and `fv` elements were set to NULL.

Author(s)

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

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

See Also

`optimsimplex.new`

Get functions	*Optimsimplex Get Function Class*

Description

The functions extract the content to various elements of a simplex object:

`optimsimplex.getall` Get all the coordinates and the function values of all the vertices.

`optimsimplex.getallfv` Get all the function values of all the vertices.

`optimsimplex.getallx` Get all the coordinates of all the vertices.

`optimsimplex.getfv` Get the function value at a given index.

`optimsimplex.getn` Get the dimension of the space of the simplex.

`optimsimplex.getnbve` Get the number of vertices of the simplex.

`optimsimplex.getve` Get the vertex at a given index in the current simplex.

`optimsimplex.getx` Get the coordinates of the vertex at a given index in the current simplex.

Usage

```
optimsimplex.getall(this = NULL)
optimsimplex.getallfv(this = NULL)
optimsimplex.getallx(this = NULL)
optimsimplex.getfv(this = NULL, ive = NULL)
optimsimplex.getn(this = NULL)
optimsimplex.getnbve(this = NULL)
optimsimplex.getve(this = NULL, ive = NULL)
optimsimplex.getx(this = NULL, ive = NULL)
```

Arguments

`this`	A simplex object.
`ive`	Vertex index.

Value

`optimsimplex.getall` Return a nbve x n+1 matrix, where n is the dimension of the space, nbve is the number of vertices and with the following content:

- `simplex[k,1]` is the function value of the vertex k, with k = 1 to nbve,
- `simplex[k,2:(n+1)]` is the coordinates of the vertex k, with k = 1 to nbve.

`optimsimplex.getallfv` Return a row vector of function values, which kˆth element is the function value for the vertex k, with k = 1 to nbve.

`optimsimplex.getallx` Return a nbve x n matrix of vertice coordinates; any given vertex is expected to be stored at row k, with k = 1 to nbve.

`optimsimplex.getfv` Return a numeric scalar.

`optimsimplex.getn` Return a numeric scalar.

optimsimplex.getnbve Return a numeric scalar.

optimsimplex.getve Return a list with a 'type' attribute set to 'T_VERTEX' and with the following elements:

- **n** The dimension of the space of the simplex.
- **x** The coordinates of the vertex at index ive.
- **fv** The value of the function at index ive.

optimsimplex.getx Return a row vector, representing the coordinates of the vertex at index ive.

Author(s)

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

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

See Also

optimsimplex.new

Simplex gradient *Simplex Gradient*

Description

optimsimplex.gradientfv determines the simplex gradient of the function which is computed by the secondary functions optimsimplex.gradcenter and optimsimplex.gradforward.

Usage

```
optimsimplex.gradientfv(this = NULL, fun = NULL, method = "forward",
                        data = NULL)
optimsimplex.gradcenter(this = NULL, fun = NULL, data = NULL)
optimsimplex.gradforward(this = NULL)
```

Arguments

this An simplex object

fun The function to compute at vertices. The function is expected to have the following input and output arguments:

```
myfunction <- function(x, this){
...
return(list(f=f,this=this))
}
```

where x is a row vector and this a user-defined data, i.e. the data argument.

method The method used to compute the simplex gradient. Two methods are available:

	'forward' and 'centered'. The 'forward' method uses the current simplex to compute the gradient (using `optimsimplex.dirmat` and `optimsimplex.deltafv`). The 'centered' method creates an intermediate simplex and computes the average.
`data`	A user-defined data passed to the function. If data is provided, it is passed to the callback function both as an input and output argument. `data` may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

Value

`optimsimplex.gradientfv` returns a list with the following elements:

g A column vector of function gradient (with length `this$n`).

data The updated user-defined data.

`optimsimplex.gradcenter` returns a list with the following elements:

g A column vector of function gradient (with length `this$n`).

data The updated user-defined data.

`optimsimplex.gradforward` returns a column vector of function gradient (with length `this$n`).

Author(s)

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

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

See Also

`optimsimplex.new`, `optimsimplex.dirmat`, `optimsimplex.deltafv`

`optimsimplex.log` *Optimsimplex Logging*

Description

This function prints a message to screen (or log file).

Usage

```
optimsimplex.log(this = NULL, msg = NULL)
```

Arguments

`this`	An simplex object.
`msg`	A message to print.

Value

Do not return any value but print `msg` to screen if the `verbose` in `this` is set to 1.

Author(s)

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

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

See Also

`optimsimplex.new`

`optimsimplex.new` *Creates a Simplex Object*

Description

`optimsimplex.new` creates a simplex list object which contains, among other elements, a matrix of vertices and a vector of function values calculated at those vertices. The object is actually created by a secondary function based upon the value of the `method` argument:

NULL -> `optimsimplex.coords`

'axes' -> `optimsimplex.axes`

'pfeffer' -> `optimsimplex.pfeffer`

'randbounds' -> `optimsimplex.randbounds`

'spendley' -> `optimsimplex.spendley`

'oriented' -> `optimsimplex.oriented`

Usage

```
optimsimplex.new(coords = NULL, fun = NULL, data = NULL, method = NULL,
                 x0 = NULL, len = NULL, deltausual = NULL, deltazero = NULL,
                 boundsmax = NULL, boundsmin = NULL, nbve = NULL,
                 simplex0 = NULL)
optimsimplex.coords(coords = NULL, fun = NULL, data = NULL)
optimsimplex.axes(x0 = NULL, fun = NULL, len = NULL, data = NULL)
optimsimplex.pfeffer(x0 = NULL, fun = NULL, deltausual = NULL,
                     deltazero = NULL, data = NULL)
optimsimplex.randbounds(x0 = NULL, fun = NULL, boundsmin = NULL,
                        boundsmax = NULL, nbve = NULL, data = NULL)
optimsimplex.spendley(x0 = NULL, fun = NULL, len = NULL, data = NULL)
optimsimplex.oriented(simplex0 = NULL, fun = NULL, data = NULL)
```

Arguments

`coords` The matrix of point estimate coordinates in the simplex. The coords matrix is expected to be a nbve x n matrix, where n is the dimension of the space and nbve is the number of vertices in the simplex, with nbve>= n+1. Only used if `method` is set to NULL.

`fun` The function to compute at vertices. The function is expected to have the following input and output arguments:

```
myfunction <- function(x, this){
 ...
 return(list(f=f,this=this))
 }
```

where x is a row vector and this a user-defined data, i.e. the `data` argument.

`data` A user-defined data passed to the function. If data is provided, it is passed to the callback function both as an input and output argument. `data` may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

`method` The method used to create the new optimsimplex object, either 'axes', 'pfeffer', 'randbounds', 'spendley' or 'oriented'.

`x0` The initial point estimates, as a row vector of length n.

`len` The dimension of the simplex. If length is a value, that unique length is used in all directions. If length is a vector with n values, each length is used with the corresponding direction. Only used if `method` is set to 'axes' or 'spendley'.

`deltausual` The absolute delta for non-zero values. Only used if `method` is set to 'pfeffer'.

`deltazero` The absolute delta for zero values. Only used if `method` is set to 'pfeffer'.

`boundsmin` A vector of minimum bounds. Only used if `method` is set to 'randbounds'.

`boundsmax` A vector of maximum bounds. Only used if `method` is set to 'randbounds'.

`nbve` The total number of vertices in the simplex. Only used if `method` is set to 'randbounds'.

`simplex0` The initial simplex. Only used if `method` is set to 'oriented'.

Details

All arguments of `optimsimplex.new` are optional. If no input is provided, the new simplex object is empty.

If `method` is NULL, the new simplex object is created by `optimsimplex.coords`. If `coords` is NULL, the simplex object is empty; otherwise, `coords` is used as the initial vertice coordinates in the new simplex.

If `method` is set to 'axes', the new simplex object is created by `optimsimplex.axes`. The initial vertice coordinates are stored in a nbve x n matrix built as follows:

```
[,1]    | x0[1] .... x0[n] |    | len[1] ... 0 |
```

```
[,.]       |   ...    ...   ...    |  +  |   ...   ...   ...    |
[,nbve]    |  x0[1]   ...  x0[n]   |     |    0    ...  len[n]  |
```

If `method` is set to 'pfeffer', the new simplex object is created by `optimsimplex.pfeffer` using the Pfeffer's method, i.e. a relative delta for non-zero values and an absolute delta for zero values.

If `method` is set to 'randbounds', the new simplex object is created by `optimsimplex.randbounds`. The initial vertice coordinates are stored in a nbve x n matrix consisting of the initial point estimates (on the first row) and a (nbve-1) x n matrix of randomly sampled numbers between the specified the bounds. The number of vertices `nbve` in the simplex is arbitrary.

If `method` is set to 'spendley', the new simplex object is created by `optimsimplex.spendley` using the Spendely's method, i.e. a regular simplex made of nbve = n+1 vertices.

If `method` is set to 'oriented', the new simplex object is created by `optimsimplex.oriented` in sorted order. The new simplex has the same sigma- length of the base simplex, but is "oriented" depending on the function value. The created simplex may be used, as Kelley suggests, for a restart of Nelder-Mead algorithm.

Value

Return a list with the following elements:

newobj A list with a 'type' attribute set to 'T_SIMPLEX' and with 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.
- **n** The dimension of the space.
- **fv** The values of the function at given vertices. It is a column matrix of length nbve.
- **nbve** The number of vertices.

data The updated `data` input argument.

Author(s)

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

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

References

"A Simplex Method for Function Minimization", Nelder, J. A. and Mead, R. The Computer Journal, January, 1965, 308-313

"Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation", W. Spendley, G. R. Hext, F. R. Himsworth, Technometrics, Vol. 4, No. 4 (Nov., 1962), pp. 441-461, Section 3.1

"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

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

"Multi-Directional Search: A Direct Search Algorithm for Parallel Machines", by E. Boyd, Kenneth W. Kennedy, Richard A. Tapia, Virginia Joanne Torczon, Virginia Joanne Torczon, 1989, Phd Thesis, Rice University

"Grid Restrained Nelder-Mead Algorithm", Arpad Burmen, Janez Puhan, Tadej Tuma, Computational Optimization and Applications, Volume 34 , Issue 3 (July 2006), Pages: 359 - 375

"A convergent variant of the Nelder-Mead algorithm", C. J. Price, I. D. Coope, D. Byatt, Journal of Optimization Theory and Applications, Volume 113 , Issue 1 (April 2002), Pages: 5 - 19,

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

Examples

```
myfun <- function(x,this){return(list(f=sum(x^2),this=this))}
mat <- matrix(c(0,1,0,0,0,1),ncol=2)

optimsimplex.new()
optimsimplex.new(coords=mat,x0=1:4,fun=myfun)
optimsimplex.new(method='axes',x0=1:4,fun=myfun)
optimsimplex.new(method='pfeffer',x0=1:6,fun=myfun)
opt <- optimsimplex.new(method='randbounds',x0=1:6,boundsmin=rep(0,6),
                        boundsmax=rep(10,6),fun=myfun)
opt
optimsimplex.new(method='spendley',x0=1:6,fun=myfun,len=10)
optimsimplex.new(method='oriented',simplex=opt$newobj,fun=myfun)
```

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

Description

The goal of this package is to provide a building block for optimization algorithms based on a simplex. The **optimsimplex** package may be used in the following optimization methods:

- the simplex method of Spendley et al.,
- the method of Nelder and Mead,
- the Box's algorithm for constrained optimization,
- the multi-dimensional search by Torczon,
- etc ...

Features The following is a list of features currently provided:

- Manage various simplex initializations
  - initial simplex given by user,
  - initial simplex computed with a length and along the coordinate axes,
  - initial regular simplex computed with Spendley et al. formula,

    - initial simplex computed by a small perturbation around the initial guess point,
    - initial simplex computed from randomized bounds.
- sort the vertices by increasing function values,
- compute the standard deviation of the function values in the simplex,
- compute the simplex gradient with forward or centered differences,
- shrink the simplex toward the best vertex,
- etc...

Details

Package:	optimsimplex
Type:	Package
Version:	1.0-2
Date:	2010-05-11
License:	CeCILL-2
LazyLoad:	yes

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

Author(s)

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

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

`optimsimplex.print` *Simplex Formatting and Display*

Description

`optimsimplex.tostring` formats the coordinates and function values in a character vector.

`optimsimplex.print` displays to screen the content of the current simplex with dimensions, coordinates and function values. This function calls `optimsimplex.tostring` to format the content of the simplex.

Usage

```
optimsimplex.print(this = NULL)
optimsimplex.tostring(this = NULL)
```

Arguments

`this`	A simplex object.

Value

`optimsimplex.tostring` returns a vector of character string of length nbve, where nbve is the number of vertices.

`optimsimplex.print` does not return any value but print to screen (or log file) the content of the current simplex.

Author(s)

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

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

See Also

`optimsimplex.new`

Examples

```
opt <- optimsimplex.new(method='axes',x0=1:5)$newobj
optimsimplex.tostring(opt)
optimsimplex.print(opt)
```

`optimsimplex.reflect` *Simplex Reflection*

Description

This function returns a new simplex by reflection of the current simplex with respect to the first vertex in the simplex. This move is used in the centered simplex gradient.

Usage

```
optimsimplex.reflect(this = NULL, fun = NULL, data = NULL)
```

Arguments

`this` An simplex object.

`fun` The function to compute at vertices. The function is expected to have the following input and output arguments:

```
myfunction <- function(x, this){
 ...
 return(list(f=f,this=this))
 }
```

where x is a row vector and this a user-defined data, i.e. the `data` argument.

`data` A user-defined data passed to the function. If data is provided, it is passed to the callback function both as an input and output argument. `data` may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

Value

Return a list with the following elements:

r The reflected simplex object.

data The updated user-defined data.

Author(s)

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

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

See Also

`optimsimplex.new`

`Set functions` *Optimsimplex Set Function Class*

Description

The functions assign content to various elements of a simplex object:

`optimsimplex.setall` Set all the coordinates and the function values of all the vertices.

`optimsimplex.setallfv` Set all the function values of all the vertices.

`optimsimplex.setallx` Set all the coordinates of all the vertices.

`optimsimplex.setfv` Set the function value at a givenindex.

`optimsimplex.setn` Set the dimension of the space of the simplex.

`optimsimplex.setnbve` Set the number of vertices of the simplex.

`optimsimplex.setve` Set the coordinates of the vertex and the function values at a given index in the current simplex.

`optimsimplex.setx` Set the coordinates of the vertex at a given index in the current simplex.

Usage

```
optimsimplex.setall(this = NULL, simplex = NULL)
optimsimplex.setallfv(this = NULL, fv = NULL)
optimsimplex.setallx(this = NULL, x = NULL)
optimsimplex.setfv(this = NULL, ive = NULL, fv = NULL)
optimsimplex.setn(this = NULL, n = NULL)
optimsimplex.setnbve(this = NULL, nbve = NULL)
optimsimplex.setve(this = NULL, ive = NULL, fv = NULL, x = NULL)
optimsimplex.setx(this = NULL, ive = NULL, x = NULL)
```

Arguments

`this` A simplex object.

`simplex` The simplex to set. It is expected to be a nbve x n+1 matrix where n is the dimension of the space, nbve is the number of vertices and with the following content:

- `simplex[k,1]` is the function value of the vertex k, with k = 1 to nbve,
- `simplex[k,2:(n+1)]` is the coordinates of the vertex k, with k = 1 to nbve.

`fv` A row vector of function values; `fv[k]` is expected to be the function value for the vertex k, with k = 1 to nbve. For `optimsimplex.setfv`, `fv` is expected to be a numerical scalar.

`x` The nbve x n matrix of vertice coordinates; the vertex is expected to be stored in `x[k,1:n]`, with k = 1 to nbve. For `optimsimplex.setve` and `optimsimplex.setx`, `x` is expected to be a row matrix.

`ive` Vertex index.

`n` The dimension of the space of the simplex.

`nbve` The number of vertices of the simplex.

Value

Return a updated simplex object `this`.

Author(s)

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

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

See Also

`optimsimplex.new`

`optimsimplex.shrink` *Simplex Shrink*

Description

This function shrinks the simplex with given coefficient sigma and returns an updated simplex. The shrink is performed with respect to the first point in the simplex.

Usage

```
optimsimplex.shrink(this = NULL, fun = NULL, sigma = 0.5, data = NULL)
```

Arguments

`this` An simplex object

`fun` The function to compute at vertices. The function is expected to have the following input and output arguments:

```
myfunction <- function(x, this){
 ...
 return(list(f=f,this=this))
 }
```

where x is a row vector and this a user-defined data, i.e. the `data`.

`sigma` The shrinkage coefficient. The default value is 0.5.

`data` A user-defined data passed to the function. If data is provided, it is passed to the callback function both as an input and output argument. `data` may be used if the function uses some additionnal parameters. It is returned as an output parameter because the function may modify the data while computing the function value. This feature may be used, for example, to count the number of times that the function has been called.

Value

Return a list with the following elements:

this The updated simplex object.

data The updated user-defined data.

Author(s)

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

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

See Also

`optimsimplex.new`

`optimsimplex.utils` *Optimsimplex Utility Functions*

Description

These functions enable various calculations and checks on the current simplex:

`optimsimplex.center` Compute the center of the current simplex.

`optimsimplex.check` Check the consistency of the data in the current simplex.

`optimsimplex.deltafv` Compute the vector of function value differences with respect to the function value at the first vertex (the lowest).

`optimsimplex.deltafvmax` Compute the difference of function value between the lowest and the highest vertices. It is expected that the first vertex (`this$x[1,]`) is associated with the smallest function value and that the last vertex (`this$x[nbve,]`) is associated with the highest function value.

`optimsimplex.dirmat` Compute the matrix of simplex direction, i.e. the matrix of differences of vertice coordinates with respect to the first vertex.

`optimsimplex.fvmean` Compute the mean of the function values in the current simplex.

`optimsimplex.fvstdev` Compute the standard deviation of the function values in the current simplex.

`optimsimplex.fvvariance` Compute the variance of the function values in the current simplex.

`optimsimplex.size` Determines the size of the simplex.

`optimsimplex.sort` Sort the simplex by increasing order of function value, so the smallest function is at the first vertex.

`optimsimplex.xbar` Compute the center of n vertices, by excluding the vertex with index `iexcl`. The default of `iexcl` is the number of vertices: in that case, if the simplex is sorted in increasing function value order, the worst vertex is excluded.

Usage

```
optimsimplex.center(this = NULL)
optimsimplex.check(this = NULL)
optimsimplex.deltafv(this = NULL)
optimsimplex.deltafvmax(this = NULL)
optimsimplex.dirmat(this = NULL)
optimsimplex.fvmean(this = NULL)
optimsimplex.fvstdev(this = NULL)
optimsimplex.fvvariance(this = NULL)
optimsimplex.size(this = NULL, method = NULL)
optimsimplex.sort(this = NULL)
optimsimplex.xbar(this = NULL, iexcl = NULL)
```

Arguments

`this` The current simplex.

`method` The method to use to compute the size of the simplex. The available methods are the following:

- **'sigmaplus'** (this is the default) The sigmamplus size is the maximum 2-norm length of the vector from each vertex to the first vertex. It requires one loop over the vertices.
- **'sigmaminus'** The sigmaminus size is the minimum 2-norm length of the vector from each vertex to the first vertex. It requires one loop over the vertices.
- **'Nash'** The 'Nash' size is the sum of the norm of the norm-1 length of the vector from the given vertex to the first vertex. It requires one loop over the vertices.
- **'diameter'** The diameter is the maximum norm-2 length of all the edges of the simplex. It requires 2 nested loops over the vertices.

`iexcl` The index of the vertex to exclude in center computation.

Value

`optimsimplex.center` Return a vector of length nbve, where nbve is the number of vertices in the current simplex.

`optimsimplex.check` Return an error message if the dimensions of the various elements of the current simplex do not match.

`optimsimplex.deltafv` Return a column vector of length nbve-1.

`optimsimplex.deltafvmax` Return a numeric scalar.

`optimsimplex.dirmat` Return a n x n numeric matrix, where n is the dimension of the space of the simplex.

`optimsimplex.fvmean` Return a numeric scalar.

`optimsimplex.fvstdev` Return a numeric scalar.

`optimsimplex.fvvariance` Return a numeric scalar.

`optimsimplex.size` Return a numeric scalar.

`optimsimplex.sort` Return an updated simplex object.

`optimsimplex.xbar` Return a row vector of length n.

Author(s)

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

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

References

"Compact Numerical Methods For Computers - Linear Algebra and Function Minimization", J.C. Nash, 1990, Chapter 14. Direct Search Methods

"Iterative Methods for Optimization", C.T. Kelley, 1999, Chapter 6., section 6.2

See Also

`optimsimplex.new`

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.