The optimbase Package - version 1.0-8

Sébastien Bihorel

March 3, 2012

optimbase 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 optimbase module.

Currently, **optimbase** does not include all functions distributed with the original Scilab module but only those required for the proper operation of the `fminsearch` function from the **neldermead** package.

1 Overview

1.1 Description

The goal of this package is to provide a building block for a large class of specialized optimization methods. This package manages the number of variables, the minimum and maximum bounds, the number of non linear inequality constraints, the logging system, various termination criteria, the cost function, etc...

The optimization problem to solve is the following:

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

where n is the number of variables and $nbineq$ the number of inequality constraints.

1.2 Basic object

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

1.3 The cost function

The `fun` element of the optimization object (thereafter referred to as `this`) allows to configure the cost function. The cost function is used, depending on the context, to compute the cost, the non-linear inequality positive constraints, the gradient of the function and the gradient of the nonlinear inequality constraints. The cost function can also be used to produce outputs and to terminate an optimization algorithm. The cost function can also take as input/output an additional argument, if

the `costfargument` element of `this` is configured. It should be defined as follows:

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

where

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

`index`: an integer representing the value to compute:

- index = 1: nothing is to be computed, the user may display messages, for example
- index = 2: compute `f`
- index = 3: compute `g`
- index = 4: compute `f` and `g`
- index = 5: compute `c`
- index = 6: compute `f` and `c`
- index = 7: compute `f`, `g`, `c` and `gc`

where `f` is the value of the objective function (a scalar), `g` the gradient of the objective function (a row matrix), `c` the constraints (a row matrix), and `gc` the gradient of the constraints (a matrix),

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

The cost function must return a list with the following elements: `this`, `f`, `g`, `c`, `gc`, `index`. The `index` output parameter has a different meaning than the `index` input argument; it indicates if the evaluation of the cost function was possible:

- index > 0: everything went fine,
- index $= 0$: the optimization must stop,
- index < 0: one function could not be evaluated.

The cost function is typically evaluated at the current point estimate `x` by using the following call: `optimbase.function(this, x, index)`.

If the 'type' attribute of `this$costfargument` is **not** 'T_FARGS', the cost function is called within the `optimbase.function` as `this$fun(x=x,index=index)` and returns non NULL elements for:

- `f`, and `index`: if `this$withderivatives` is FALSE and `this$nbineqconst=0` (there is no nonlinear constraint),
- `f`, `c`, and `index`: if `this$withderivatives` is FALSE and `this$nbineqconst>0` (there are nonlinear constraints),
- `f`, `g`, and `index`: if `this$withderivatives` is TRUE and `this$nbineqconst=0` (there is no nonlinear constraint),
- `f`, `g`, `c`, `gc`, and `index`: if `this$withderivatives` is TRUE and `this$nbineqconst>0` (there are nonlinear constraints).

If the 'type' attribute of `this$costfargument` is 'T_FARGS', the cost function is called within the `optimbase.function` as `this$fun(x=x,index=index,fmsfundata=this$costfargument)` and returns non NULL elements for:

- `f`, `index`, and `this$costfargument`: if `this$withderivatives` is FALSE and `this$nbineqconst=0` (there is no nonlinear constraint),
- `f`, `c`, `index`, and `this$costfargument`: if `this$withderivatives` is FALSE and `this$nbineqconst>0` (there are nonlinear constraints),
- `f`, `g`, `index`, and `this$costfargument`: if `this$withderivatives` is TRUE and `this$nbineqconst=0` (there is no nonlinear constraint),
- `f`, `g`, `c`, `gc`, `index`, and `this$costfargument`: if `this$withderivatives` is TRUE and `this$nbineqconst>0` (there are nonlinear constraints).

Each of these cases corresponds to a particular class of algorithms, including for example unconstrained, derivative-free algorithms, nonlinearily constrained, derivative-free algorithms, unconstrained, derivative-based algorithms, nonlinearily constrained, derivative-based algorithms, etc... The current package was designed to handle many situations.

1.4 The output function

The `outputcommand` element of the optimization 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. Possible values are 'init', 'iter', and 'done'.

data: a list containing at least the following elements:

x: the current point estimate,

fval: the value of the cost function at the current point estimate,

iteration: the current iteration index,

funccount: the number of function evaluations.

fmsdata: a user-defined parameter. This input parameter is defined with the `outputcommandarg` element of the optimization 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 `optimbase.terminate` function provided with the current package takes into account several generic termination criteria. It is recommended that specialized termination criteria in specialized optimization algorithms are implemented by calling extra termination criteria function in addition to the `optimbase.terminate`, rather than by modification of the function itself.

The `optimbase.terminate` function uses a set of rules to determine whether the algorithm should continue or stop. It also updates the termination status to one of the following: 'continue', 'maxiter', 'maxfunevals', 'tolf' or 'tolx'. The set of rules is the following:

- 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 optimization 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` element of the optimization object: 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` element of the optimization object:

  FALSE: the tolerance on `f` is just skipped.

  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`.
  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 of the optimization object:

  FALSE: the tolerance on `x` is just skipped.

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

  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.

2 Network of optimbase functions

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

3 Help on optimbase functions

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

Description

The goal of this package is to provide a building block for a large class of specialized optimization methods. This packages manages:

- the number of variables,
- the minimum and maximum bounds,
- the number of non linear inequality constraints,
- the cost function,
- the logging system,
- various termination criteria,
- etc...

Features The following is a list of features the optimbase toolbox currently provided:

- 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 the vector of estimated parameter `x` (relative or absolute),
  - maximum number of evaluations of the cost function,
- Manage the history of the convergence, including:
  - history of function values,
  - history of optimum point.
- Provide query features for
  - the status of the optimization process,
  - the number of iterations,
  - the number of function evaluations,
  - function value at initial point,
  - function value at optimal point,
  - the optimum parameters,
  - etc...

Details

Package:	optimbase
Type:	Package
Version:	1.0-8
Date:	2012-03-03
License:	CeCILL-2
LazyLoad:	yes

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

Author(s)

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

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

`asserts` *Check of Variable Class*

Description

Utility functions in **optimbase** meant to check variable class. Stop the algorithm if the variable is not of the expected class.

`assert.typeboolean` for logical variables

`assert.typefunction` for functions

`assert.typereal` for numeric variables

`assert.typestring` for character variables

`unknownValueForOption` stops the algorithm and returns an error message, when some checks in `optimbase` are not successful.

Usage

```
assert.typeboolean(var = NULL, varname = NULL, ivar = NULL)
assert.typefunction(var = NULL, varname = NULL, ivar = NULL)
assert.typereal(var = NULL, varname = NULL, ivar = NULL)
assert.typestring(var = NULL, varname = NULL, ivar = NULL)
unknownValueForOption(value = NULL, optionname = NULL)
```

Arguments

`var`	The variable name.
`varname`	The name of a variable to which `var` should have been assigned to.
`ivar`	A integer, meant to provide additional info on `varname` in the error message.
`value`	A numeric or a string.
`optionname`	The name of a variable for which `value` is unknown.

Value

Return an error message through the `stop` function.

Author(s)

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

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

`zeros & ones` *Matrix of zeros or ones.*

Description

Creates a matrix of zeros or ones.

Usage

```
zeros(nx = 1, ny = nx)
ones(nx = 1, ny = nx)
```

Arguments

`nx`	The number of rows. Default is 1.
`ny`	The number of columns. Default is `nx`.

Details

`zeros` and `ones` create full matrices of zeros and ones. If the user only provides an input for `nx`, the produced matrices are nx x nx square matrices.

Value

Return of nx x ny matrix of zeros of ones.

Author(s)

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

Examples

```
zeros()
zeros(3)
ones(4,5)
## Not run: ones('3','3')
```

`optimbase.checkbounds` *Check bounds.*

Description

This function checks if the bounds defined in the optimization object are consistent (same number of minimal and maximal bounds as the number of variables, minimal bounds lower than maximal bounds) and puts an error message in the returned object if not.

Usage

```
optimbase.checkbounds(this = NULL)
```

Arguments

`this`	An optimization object.

Value

Return a list with the following list:

this The optimization object.

isok TRUE if the bounds are consistent, FALSE otherwise.

errmsg An error message if the bounds are not consistent.

Author(s)

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

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

`optimbase.checkcostfun` *Check Cost Function*

Description

This function checks that the cost function is correctly specified in the optimization object, including that the elements of `this` used by the cost function are consistent.

Usage

```
optimbase.checkcostfun(this = NULL)
```

Arguments

`this`	An optimization object

Details

Depending on the definition of nonlinear constraints (`nbineqconst` element > 0) and the use of derivatives (`withderivatives` element set to TRUE), this function makes several cost function calls with different `index` value (see `vignette('optimbase',package='optimbase')` for more details about `index`). If at least one call fails, the function stops the search algorithm.

Following every successful cost function call, `optimbase.checkcostfun` calls `optimbase.checkshape` to check the dimensions of the matrix returned by the cost function against some expectations.

Value

Return the optimization object or an error message if one check is not successful.

Author(s)

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

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

See Also

`optimbase.checkshape`

`optimbase.checkshape` *Check the Dimensions of the Cost Function Output*

Description

This function is called by `optimbase.checkcostfun` to check whether the dimensions of a cost function output match the expectations.

Usage

```
optimbase.checkshape(this = NULL, varname = NULL, data = NULL, index = NULL,
                     expectednrows = NULL, expectedncols = NULL)
```

Arguments

`this`	An optimization object.
`varname`	The name of the output being checked, either 'f', 'c', or 'g'.
`data`	A content of the output.
`index`	The index (see `vignette('optimbase',package='optimbase')` for more details).
`expectednrows`	Number of expected rows.
`expectedncols`	Number of expected columns.

Value

Return the optimization object or an error message if the dimensions are inconsistent.

Author(s)

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

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

See Also

`optimbase.checkcostfun`

`optimbase.checkx0` *Check Consistency of Initial Guesses*

Description

This function checks that the initial guesses defined in the optimization object are consistent with the defined bounds and the non linear inequality constraints. The actual work is delegated to `optimbase.isfeasible`.

Usage

```
optimbase.checkx0(this = NULL)
```

Arguments

`this`	An optimization object

Value

Return a list with the following elements:

this The optimization object.

isok TRUE if the initial guesses are consistent with the settings, FALSE otherwise.

Author(s)

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

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

See Also

`optimbase.isfeasible`

`optimbase.configure` *Optimization Object Configuration*

Description

This functions configures the current optimization object with the given `value` for the given `key`.

Usage

```
optimbase.configure(this = NULL, key = NULL, value = NULL)
optimbase.set(this = NULL, key = NULL, value = NULL)
optimbase.histset(this = NULL, iter = NULL, key = NULL, value = NULL)
```

Arguments

`this`	The current optimization object.
`key`	The key to configure. See details for the list of possible keys.
`value`	The value to assign to the key.
`iter`	The iteration at which the data must be stored.

Details

`optimbase.configure` and `optimbase.set` set the content of the `key` element of the optimization object `this` to `value`.

The only available keys in `optimbase.configure` are the following:

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

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

'-x0' The initial guesses, 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' (see `vignette('optimbase',package=` for more details).

'-maxiter' The maximum number of iterations. If this criteria is triggered during optimization, the status of the optimization is set to 'maxiter' (see `vignette('optimbase',package='optimbase')` for more details).

'-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 function and the non linear constraints, if any. See `vignette('optimbase',package='optimbase')` 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. Details of the communication between the optimization system and the output command function are provided in `vignette('optimbase',package='optimbase')`.

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

'-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.

'-logfile' The name of the log file.

'-withderivatives' Set to TRUE if the algorithm uses derivatives.

The only available keys in `optimbase.set` are the following:

'-iterations' the number of iterations.

'-xopt' the optimum point estimate.

'-fopt' the value of the cost function at the optimum point estimate.

'-historyxopt' a list, with nbiter element, containing the history of x during the iterations. This list is available after optimization if the history storing was enabled with the `storehistory` element.

'-historyfopt' an vector, with nbiter values, containing the history of the function value during the iterations. This vector is available after optimization if the history storing was enabled with the `storehistory` element.

'-fx0' the value of the cost function at the initial point estimate.

'-status' a string containing the status of the optimization.

The only available keys in `optimbase.histset` are '-historyxopt' and '-historyfopt'. Contrary to `optimbase.set`, this function only alters the value of `historyxopt` and `historyfopt` at the specific iteration `iter`.

Value

An updated optimization object.

Author(s)

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

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

See Also

optimbase.new

---

optimbase.destroy *Erase an optimization history.*

Description

Erase the optimization history in an optimization object.

Usage

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

Arguments

this	An optimization object.

Details

This function erases the content of the historyfopt and historyxopt elements in this and call the optimbase.logshutdown function if the logstartup element in this is set to TRUE.

Value

Return an updated optimization object.

Author(s)

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

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

See Also

optimbase.logshutdown

`optimbase.function` *Call Cost Function*

Description

This function calls the cost function defined in the `fun` element of the current object and returns the required results. If an additionnal argument for the cost function is defined in current object, it is passed to the function as the last argument. See `vignette('optimbase',package='optimbase')` for more details.

Usage

```
optimbase.function(this = NULL, x = NULL, index = NULL)
```

Arguments

`this`	An optimization object.
`x`	The point estimate where the cost function should be evaluated, i.e. a column vector.
`index`	An integer between 1 and 6 (see `vignette('omptimbase',package='optimbase')` for more details).

Value

Return a list with the following elements:

this The updated optimization object.

f The value of the cost function.

g The gradient of the cost function.

c The nonlinear, positive, inequality constraints.

gc The gradient of the nonlinear, positive, inequality constraints.

index An integer:

- if index > 0, everything went fine,
- if index $== 0$, interrupts the optimization,
- if index < 0, one of the function could not be evaluated.

Author(s)

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

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

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

Description

Get the value for the given key in an optimization object.

Usage

```
optimbase.get(this = NULL, key = NULL)
optimbase.cget(this = NULL, key = NULL)
optimbase.histget(this = NULL, iter = NULL, key = NULL)
```

Arguments

`this`	An optimization object.
`key`	The name of the key to quiery. The list of available keys for query with `optimbase.get` is: '-funevals', '-iterations', '-xopt', '-fopt', '-historyxopt', '-historyfopt', '-fx0', '-status', and '-logstartup'. The list of available keys for query with `optimbase.cget` is: '-verbose', '-verbosetermination', '-function', '-method', '-x0', '-maxfunevals', '-maxiter', '-tolfunabsolute', '-tolfunrelative', '-tolxabsolute', '-tolxrelative', '-tolxmethod', '-tolfunmethod', '-outputcommand', '-outputcommandarg', '-numberofvariables', '-storehistory', '-costfargument', '-boundsmin', '-boundsmax', '-nbineqconst', '-logfile', and '-withderivatives'. The list of available keys for query with `optimbase.histget` is: '-historyxopt' and '-historyfopt'.
`iter`	The iteration at which the data is stored.

Details

`optimbase.get` extracts the value of elements which are not available directly to the user interface, but are computed internally, while `optimbase.cget` extracts the value of elements which are available to the user interface. While `optimbase.get` extracts the entire content of `historyxopt` and `historyfopt`, `optimbase.histget` only extracts the content of the history at the iteration `iter`.

Value

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

Author(s)

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

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

See Also

optimbase.new, optimbase.configure

---

optimbase.gridsearch *Grid evaluation of a constrained or unconstrained cost function*

Description

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

Usage

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

Arguments

fun	A constrained or unconstrained cost function defined as described in the vignette (vignette('optimbase',package='optimbase')).
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 numbers greater than 1, 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

optimbase.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].

optimbase.gridsearch also determines if the cost function is feasible at each evaluation point by calling optimbase.isfeasible.

Value

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

Author(s)

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

See Also

optimbase.isfeasible

Examples

```
# Problem: find x and y that maximize 3.6*x - 0.4*x^2 + 1.6*y - 0.2*y^2 and
#          satisfy the constrains:
#            2*x - y <= 10
#            x >= 0
#            y >= 0
#

gridfun <- function(x=NULL,index=NULL,fmsfundata=NULL,...){

  f <- c()
  c <- c()
  if (index == 2 | index == 6)
    f <- -(3.6*x[1] - 0.4*x[1]*x[1] + 1.6*x[2] - 0.2*x[2]*x[2])
  if (index == 5 | index == 6)
    c <- c(10 - 2*x[1] - x[2],
           x[1],
           x[2])
  varargout <- list(f = f, g = c(), c = c, gc = c(), index = index,
                    this = list(costfargument = fmsfundata))

  return(varargout)
}


x0 <- c(0.35,0.3)
npts <- 6
alpha <- 10

optimbase.gridsearch(fun=gridfun,x0=x0,xmin=NULL,xmax=NULL,
                     npts=npts,alpha=alpha)

# 3.5 and 3 is the actual solution of the optimization problem
```

`Bounds & constraints` *Query for Bounds and Constraints*

Description

`optimbase.hasbounds` and `optimbase.hascons` query an optimization object and determine whether bounds and nonlinear constraints have been specified. Bounds are defined in the `boundsmin` and `boundsmax` elements of the optimization object. The number of nonlinear constraints is defined in the `nbineqconst` element.

`optimbase.hasconstraints` determine whether any bound or constraint has been specified.

Usage

```
optimbase.hasbounds(this = NULL)
optimbase.hasnlcons(this = NULL)
optimbase.hasconstraints(this = NULL)
```

Arguments

`this`	An optimization object.

Value

Return TRUE if bounds or constraints are found, FALSE otherwise.

Author(s)

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

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

`optimbase.incriter` *Iteration Log Incrementation*

Description

This function increments the number of iterations stored in the `iterations` element of the optimization object.

Usage

```
optimbase.incriter(this = NULL)
```

Arguments

`this`	An optimization object.

Value

Return the optimization object after increasing the content of the `iterations` element by 1 unit.

Author(s)

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

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

---

optimbase.isfeasible *Check Point Estimate*

Description

This function checks that the point estimate is consistent with the bounds and the non linear inequality constraints. It is usually called by `optimbase.checkx0` to check initial guesses.

Usage

```
optimbase.isfeasible(this = NULL, x = NULL)
```

Arguments

`this`	An optimization object.
`x`	The point estimate, i.e. a column vector of numerical values.

Details

Returns 1 if the given point satisfies bounds constraints and inequality constraints.

Returns 0 if the given point is not in the bounds.

Returns -1 if the given point does not satisfies inequality constraints.

Value

Return a list with the following elements:

this The optimization object.

isfeasible The feasibility flag, either -1, 0 or 1.

Author(s)

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

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

See Also

`optimbase.checkx0`

Bound and constraint checks

Point Estimate Comparison with Bounds and Constraints

Description

`optimbase.isinbounds` checks that given parameter estimates are within the defined minimum and maximum boundaries, while `optimbase.isinnonlincons` checks that the given point estimate satisfies the defined nonlinear constraints.

Usage

```
optimbase.isinbounds(this = NULL, x = NULL)
optimbase.isinnonlincons(this=NULL,x=NULL)
```

Arguments

`this`	An optimization object.
`x`	A column vector of parameter estimates.

Value

Both functions return a list with the following elements:

this The optimization object.

isfeasible TRUE if the parameter estimates satisfy the constraints, FALSE otherwise.

Author(s)

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

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

Log functions

Optimbase Log functions

Description

`optimbase.logstartup` initializes logging if verbose logging is enabled (via the `verbose` element of the optimization object). If the logging has already been initialized, it generates an error and stops the optimization.

If verbose logging is enabled, `optimbase.log` prints the given message in the console. If verbose logging is disabled, it does nothing. If the `logfile` element of the optimization object has been set, it writes the message into the file instead of writing to the console.

`optimbase.stoplog` prints the given stopping rule message if verbose termination is enabled (via the `verbosetermination` element of the optimization object). If verbose termination is disabled, it does nothing.

`optimbase.logshutdown` turns verbose logging off.

Usage

```
optimbase.logstartup(this = NULL)
optimbase.log(this = NULL, msg = NULL)
optimbase.stoplog(this = NULL, msg = NULL)
optimbase.logshutdown(this = NULL)
```

Arguments

`this`	The optimization object.
`msg`	The message to print.

Value

All functions return the unchanged optimization object.

Author(s)

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

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

`optimbase.outputcmd` *Call user-defined output function*

Description

Call user-defined output function.

Usage

```
optimbase.outputcmd(this = NULL, state = NULL, data = NULL)
```

Arguments

`this`	An optimization object.
`state`	The current state of the algorithm: either 'init', 'iter', or 'done'.
`data`	A list containing at least the following elements: **x** the current point estimate, **fval** the value of the cost function at the current point estimate, **iteration** the current iteration index, **funccount** the number of function evaluations.

Details

The `data` list argument may contain more levels than those presented above. These additional levels 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...

Value

Do not return any data, but execute the output function defined in the `outputcommand` element of `this`.

Author(s)

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

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

`optimbase.outstruct` *Create Basic Optimization Data Object*

Description

This function creates a basic optimization data object by extracting the content of specific fields of an optimization object.

Usage

```
optimbase.outstruct(this = NULL)
```

Arguments

`this`	An optimization object.

Value

Return an object of class 'optimbase.data', i.e. a list with the following elements:

x The current optimum point estimate (extracted from `this$xopt`).

fval The value of the cost function at the current optimum point estimate (extracted from `this$fopt`).

iteration The current number of iteration (extracted from `this$iterations`).

funccount The current number of function evaluations (extracted from `this$funevals`).

Author(s)

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

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

optimbase.proj2bnds *Projection of Point Estimate to Bounds*

Description

This function determines if all elements of a point estimate are within the defined bounds. In the case one or more parameter estimates are not, the function projects those to their corresponding bounds.

Usage

```
optimbase.proj2bnds(this = NULL, x = NULL)
```

Arguments

this	An optimization object.
x	A point estimate.

Value

Return a list with the following elements:

this The optimization object.

p A vector of updated paremeter estimes. The ith element of the vector is:

- x[i] if this$boundsmin[i] < x[i] < this$boundsmax[i],
- this$boundsmin[i] if x[i] <= this$boundsmin[i],
- this$boundsmax[i] if this$boundsmax[i] <= x[i].

Author(s)

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

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

optimbase.terminate *Evaluation of Termation Status*

Description

This function determines whether the optimization must continue or terminate. If the verbosetermination element of the optimization object is enabled, messages are printed detailing the termination intermediate steps. The optimbase.terminate function takes into account the number of iterations, the number of evaluations of the cost function, the tolerance on x and the tolerance on f. See the section "Termination" in vignette('optimbase',package='optimbase') for more details.

Usage

```
optimbase.terminate(this = NULL, previousfopt = NULL, currentfopt = NULL,
                    previousxopt = NULL, currentxopt = NULL)
```

Arguments

`this`	An optimization object.
`previousfopt`	The previous value of the objective function.
`currentfopt`	The current value of the objective function.
`previousxopt`	The previous value of the parameter estimate matrix.
`currentxopt`	The current value of the parameter estimate matrix.

Value

Return a list with the following elements:

this The updated optimization object.

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

status The termination status could be 'maxiter', 'maxfuneval', 'tolf' or 'tolx' if `terminate` is set to TRUE, 'continue' otherwise.

Author(s)

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

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

`optimtypeof` *Object Type or Class*

Description

This functions extracts the `type` attribute of lists commonly created by `optimbase.new`.

Usage

```
optimtypeof(object = NULL)
```

Arguments

`object`	Any object but usually a optimization, simplex or neldermead list object.

Details

If `object` is not a list or if `type` is not an attribute of `object`, `optimtypeof` returns the class of `object`.

Value

Returns a single character string which is either the content of the `type` attribute of `object` or its class.

Author(s)

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

See Also

`optimbase.new`

Examples

```
obj1 <- optimbase.new()
optimtypeof(obj1)
optimtypeof(obj1$optbase)
optimtypeof(obj1$simplex0)

obj2 <- list(1)
attr(obj2, 'type') <- 'newtype'
optimtypeof(obj2)
```

`size` *Vector, Matrix or Data.Frame Size*

Description

`size` is a utility function which determines the dimensions of vectors (coerced to matrices), matrices, arrays, data.frames, and list elements.

Usage

```
size(x = NULL, n = NULL)
```

Arguments

`x`	A R object.
`n`	A integer indicating the dimension of interest.

Details

`size` is a wrapper function around `dim`. It returns the n^th dimension of `x` if `n` is provided. If `n` is not provide, all dimensions will be determined. If `x` is a list, `n` is ignored and the dimensions of all elements of `x` are recursively determined.

Value

Returns a vector or list of dimensions.

Author(s)

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

See Also

dim

Examples

```
a <- 1
b <- letters[1:6]
c <- matrix(1:20,nrow=4,ncol=5)
d <- array(1:40, dim=c(2,5,2,2))
e <- data.frame(a,b)
f <- list(a,b,c,d,e)

size(NULL) # 0 0
size(NA)   # 1 1
size(a)    # 1 1
size(b,2)  # 6
size(c)    # 4 5
size(d)    # 2 5 2 2
size(e,3)  # NA
size(f)
```

---

strvec *Auto-collapse of Vectors*

Description

strvec is a utility function which collapses all elements of a vector into a character scalar.

Usage

```
strvec(x = NULL)
```

Arguments

x	A string of characters.

Value

A character scalar consisting of all the elements of x separated by a single white space.

Author(s)

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

Examples

```
strvec(letters[1:10])
strvec(1:10)
```

`transpose` *Vector and Matrix Transpose*

Description

`transpose` is a wrapper function around the `t` function, which tranposes matrices. Contrary to `t`, `transpose` processes vectors as if they were row matrices.

Usage

```
transpose(object = NULL)
```

Arguments

`object`	A vector or a matrix.

Value

Return a matrix which is the exact transpose of the vector or matrix `x`

Author(s)

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

See Also

`t`

Examples

```
1:6
t(1:6)
transpose(1:6)
mat <- matrix(1:15,nrow=5,ncol=3)
mat
transpose(mat)
```

`vec2matrix` *Vector to Matrix Conversion*

Description

This function converts a vector into a row matrix.

Usage

```
vec2matrix(object = NULL)
```

Arguments

`object`	A vector or a matrix.

Details

If `object` is already a matrix, `object` is not modified. If `object` is not a matrix or a vector, the algorithm is stopped.

Value

Return a row matrix.

Author(s)

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

4 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.