Package 'optimbase'

October 14, 2022

Type Package

Title R Port of the 'Scilab' Optimbase Module

Version 1.0-10

Date 2022-01-24

Description Provides a set of commands to manage an abstract optimization method. The goal 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 cost function, the logging system, various termination criteria, etc...

Depends Matrix, methods

Suggests knitr (>= 1.28), rmarkdown (>= 2.2)

License CeCILL-2

Encoding UTF-8

VignetteBuilder knitr

LazyLoad yes

NeedsCompilation no

Author Sebastien Bihorel [aut, cre],
Michael Baudin [aut]

Maintainer Sebastien Bihorel <sb.pmlab@gmail.com>

Repository CRAN

Date/Publication 2022-01-26 19:22:48 UTC

R topics documented:

`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:	1.0-10
Date:	2022-01-24
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.classboolean` for logical variables

`assert.classfunction` for functions

`assert.classreal` for numeric variables

`assert.classinteger` for integer variables

`assert.classstring` for character variables

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

Usage

```
assert.classboolean(var = NULL, varname = NULL, ivar = NULL)
assert.classfunction(var = NULL, varname = NULL, ivar = NULL)
assert.classreal(var = NULL, varname = NULL, ivar = NULL)
assert.classinteger(var = NULL, varname = NULL, ivar = NULL)
assert.classstring(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>)

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>)

`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>)

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 *S3 optimbase classes*

Description

These functions support the S3 class 'optimbase' and related S3 classes 'optimbase.outputargs' and 'optimbase.functionargs'. They are intended to either create objects of these classes, check if an object is of these classes, or coerce it to one of these classes.

Usage

```
  optimbase(verbose, x0, fx0, xopt, fopt, tolfunabsolute,
    tolfunrelative, tolfunmethod, tolxabsolute, tolxrelative, tolxmethod,
    maxfunevals, funevals, maxiter, iterations, fun, status, historyxopt,
    historyfopt, verbosetermination, outputcommand, outputcommandarg,
    numberofvariables, storehistory, costfargument, boundsmin, boundsmax,
    nbineqconst, logfile, logfilehandle, logstartup, withderivatives)

  optimbase.outputargs(...)

  optimbase.functionargs(...)

  ## S3 method for class 'optimbase'
print(x,verbose=FALSE,...)

  ## S3 method for class 'optimbase'
is(x=NULL)

  ## S3 method for class 'optimbase'
summary(object,showhistory,...)

  ## S3 method for class 'optimbase.outputargs'
is(x=NULL)

  ## S3 method for class 'optimbase.outputargs'
as(x=NULL)

  ## S3 method for class 'optimbase.functionargs'
is(x=NULL)

  ## S3 method for class 'optimbase.functionargs'
as(x=NULL)
```

Arguments

verbose	The verbose option, controlling the amount of messages.

`x0` The initial guess.

`fx0` The value of the function for the initial guess.

`xopt` The optimum parameter.

`fopt` The optimum function value.

`tolfunabsolute` The absolute tolerance on function value.

`tolfunrelative` The relative tolerance on function value.

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

`tolxabsolute` The absolute tolerance on x..

`tolxrelative` The relative tolerance on x.

`tolxmethod` Possible values: FALSE, TRUE.

`maxfunevals` The maximum number of function evaluations.

`funevals` The number of function evaluations.

`maxiter` The maximum number of iterations.

`iterations` The number of iterations.

`fun` The cost function.

`status` The status of the optimization.

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

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

`verbosetermination`
The verbose option for termination criteria.

`outputcommand` The command called back for output. This must be a valid R function accepting the following arguments:

state A character string, typically indicating the status of the algorithm.

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 An optional object of class 'optimbase.outputargs'.

`outputcommandarg`
The outputcommand argument is initialized as an empty object of class 'optimbase.outputargs' passed to the command defined in the outputcommand element of the optimbase object. This object has no required structure or content but is typically a list which may be used to provide some extra information to the output command.

`numberofvariables`
: The number of variables to optimize.

`storehistory`
: The flag which enables/disables the storing of the history.

`costfargument`
: The costf argument is initialized as an empty object of class 'optimbase.functionargs'. This object has no required structure or content but is typically a list which may be used to provide some information to the cost function'.

`boundsmin`
: Minimum bounds for the parameters.

`boundsmax`
: Maximum bounds for the parameters.

`nbineqconst`
: The number of nonlinear inequality constraints.

`logfile`
: The name of the log file.

`logfilehandle`
: The handle for the log file.

`logstartup`
: Set to TRUE when the logging is started up.

`withderivatives`
: Set to TRUE when the method uses derivatives.

`...`
: optional arguments to 'print' or 'plot' methods.

`x`
: An object of class 'optimbase'.

`object`
: An object of class 'optimbase'.

`showhistory`
: Optional logical flag, to define whether optimization history must be summarized or not.

Value

The `optimbase` function returns a new object of class 'optimbase', i.e. a list containing the following elements:

verbose Default is FALSE.

x0 Default is NULL.

fx0 Default is NULL.

xopt Default is 0.

fopt Default is 0.

tolfunabsolute Default is 0.

tolfunrelative Default is .Machine$double.eps.

tolfunmethod Default is FALSE.

tolxabsolute Default is 0.

tolxrelative Default is .Machine$double.eps.

tolxmethod Default is TRUE.

maxfunevals Default is 100.

funevals Default is 0.

maxiter Default is 100.

iterations Default is 0.

fun Default is ".

status Default is ".

historyfopt Default is NULL.

historyxopt Default is NULL.

verbosetermination Default is FALSE.

outputcommand Default is ".

outputcommandarg Default is ". If the user configures this element, it is expected to be an object of class 'optimbase.outputargs' or will be coerced to an object of class 'optimbase.outputargs'.

numberofvariables Default is 0.

storehistory Default is FALSE.

costfargument Default is ". If the user configures this element, it is expected to be an object of class 'optimbase.functionargs' or will be coerced to an object of class 'optimbase.functionargs'.

boundsmin Default is NULL.

boundsmax Default is NULL.

nbineqconst Default is 0.

logfile Default is ".

logfilehandle Default is 0.

logstartup Default is FALSE.

withderivatives Default is FALSE.

The `print.optimbase` and `is.optimbase` functions are S3 method for objects of class 'optimbase'. The `showhistory` argument can be provided to the `print.optimbase` function to indicate whether or not the history of optimization should be printed.

The `optimbase.outputargs` function returns a new object of class 'optimbase.outputargs', i.e. a list of all arguments provided by the user. The `is.optimbase.outputargs` functions are S3 method for objects of class 'optimbase.outputargs'.

The `optimbase.functionargs` function returns a new object of class 'optimbase.functionargs', i.e. a list of all arguments provided by the user. The `is.optimbase.functionargs` functions are S3 method for objects of class 'optimbase.functionargs'.

Author(s)

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

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

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.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>)

---

optimbase.get *Get the value for the given element*

Description

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

Usage

```
optimbase.get(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: 'verbose', 'x0', 'fx0', 'xopt', 'fopt', 'tolfunabsolute', 'tolfunrelative', 'tolfunmethod', 'tolxabsolute', 'tolxrelative', 'tolxmethod', 'maxfunevals', 'maxiter', 'iterations', 'function', 'status', 'historyfopt', 'historyxopt', 'verbosetermination', 'outputcommand', 'outputcommandarg', 'numberofvariables', 'storehistory', 'costfargument', 'boundsmin', 'boundsmax', 'nbineqconst', 'logfile', 'logfilehandle', 'logstartup', 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

While optimbase.get extracts the entire content of the object element, including 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, optimbase.set

---

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)

  return(varargout)
}


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

res <- 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
print(res)
```

`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`

`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.set *Optimization Object Configuration*

Description

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

Usage

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

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

'verbose' Set to 1 to enable verbose logging.

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

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

'xopt' The optimum point estimate.

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

'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'.

'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='optimbase'` for more details).

'funevals' The number of function evaluations.

'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).

'iterations' The number of iterations.

'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.

'status' A string containing the status of the optimization.

'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.

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

'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='o`

'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.

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

'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.

'logfilehandle' Set to 1 if logging has been started

'logstartup' Set to 1 if logging has been started

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

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`

---

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>)

---

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>)

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)
# Will fail
try(ones('3','3'))
```

Index