Package ‘caRamel’

May 28, 2019

Type Package

Title Automatic Calibration by Evolutionary Multi Objective Algorithm

Version 1.1

Date 2019-05-24

Author Nicolas Le Moine [aut],
Celine Monteil [aut],
Frederic Hendrickx [ctb],
Fabrice Zaoui [aut, cre]

Maintainer Fabrice Zaoui <fabrice.zaoui@edf.fr>

Depends geometry, parallel

Suggests knitr, testthat

Description Multi-objective optimizer initially developed for the calibration of hydrological models.
The algorithm is a hybrid of the MEAS algorithm (Efstratiadis and Koutsoyiannis (2005) <doi:10.13140/RG.2.2.32963.81446>) by using the directional search method based on the simplexes of the objective space
and the epsilon-NGSA-II algorithm with the method of classification of the parameter vectors archiving management by epsilon-dominance (Reed and Devireddy <doi:10.1142/9789812567796_0004>).

License GPL-3

URL https://github.com/fzao/caRamel

BugReports https://github.com/fzao/caRamel/issues

Encoding UTF-8

LazyData TRUE

RoxygenNote 6.0.1

NeedsCompilation no

VignetteBuilder knitr

Repository CRAN

Date/Publication 2019-05-28 13:40:09 UTC

R topics documented:

`caRamel-package` *caRamel optimizer*

Description

Automatic Calibration by Evolutionary Multi Objective Algorithm

Details

caRamel is a package for multi-objective optimization of complex environmental models.

The algorithm is a hybrid of the *MEAS* algorithm (Efstratiadis and Koutsoyiannis, 2005) by using the directional search method based on the simplexes of the objective space and the *epsilon-NGSA-II* algorithm with the method of classification of the parameter vectors archiving management by epsilon-dominance (Reed and Devireddy, 2004).

The main function of the package is *caRamel()*.

This function uses all the other functions of the package.

Author(s)

Fabrice Zaoui, Nicolas Le Moine, Celine Monteil (EDF R&D - LNHE)

References

Efstratiadis, A. and Koutsoyiannis, D. (2005) *The multi-objective evolutionary annealing-simplex method and its application in calibration hydrological models*, in EGU General Assembly 2005, Geophysical Research Abstracts, Vol. 7, Vienna, 04593, European Geophysical Union. doi:10.13140/RG.2.2.32963.81446.

Le Moine, N. (2009) *Description d'un algorithme génétique multi-objectif pour la calibration d'un modèle pluie-débit* (in French). Post-Doctoral Status Rep. 2, UPMC/EDF, 13 pp.

Reed, P. and Devireddy, D. (2004) *Groundwater monitoring design: a case study combining epsilon-dominance archiving and automatic parameterization for the NSGA-II*, in Coello-Coello C, editor. Applications of multi-objective evolutionary algorithms, Advances in natural computation series, vol. 1, pp. 79-100, World Scientific, New York. doi:10.1142/9789812567796_0004.

`boxes` *Box numbering for each points individual of the population*

Description

This function returns a box number for each points individual of the population

Usage

```
boxes(points, prec)
```

Arguments

`points`	: matrix of the objectives
`prec`	: (double, length = nobj) desired accuracy for the objectives (edges of the boxes)

Value

vector of numbers for the boxes. boxes[i] gives the number of the box containing points[i].

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
points <- matrix(rexp(200), 100, 2)
prec <- c(1.e-3, 1.e-3)
# Call the function
res <- boxes(points, prec)
```

`caRamel` *MAIN FUNCTION: multi-objective optimizer*

Description

Multi-objective optimizer. It requires to define a multi-objective function (func) to calibrate the model and bounds on the parameters to optimize.

Usage

```
caRamel(nobj, nvar, minmax, bounds, func, popsize, archsize, maxrun, prec,
  repart_gene = c(5, 5, 5, 5), gpp = NULL, blocks = NULL, pop = NULL,
  funcinit = NULL, objnames = NULL, listsave = NULL, write_gen = 0,
  carallel = TRUE, numcores = NULL, graph = TRUE)
```

Arguments

`nobj`	: (integer, length = 1) the number of objectives to optimize (nobj >= 2)
`nvar`	: (integer, length = 1) the number of variables
`minmax`	: (logical, length = nobj) the objective is either a minimization (FALSE value) or a maximization (TRUE value)
`bounds`	: (matrix, nrow = nvar, ncol = 2) lower and upper bounds for the variables
`func`	: the name of the objective function to optimize. Input argument is the number of parameter set (integer) in the x matrix. The function has to return a vector of at least 'nobj' values (Objectives 1 to nobj are used for optimization, values after nobj are recorded for information.).
`popsize`	: (integer, length = 1) the population size for the genetic algorithm
`archsize`	: (integer, length = 1) the size of the Pareto front
`maxrun`	: (integer, length = 1) the max. number of simulations allowed
`prec`	: (double, length = nobj) the desired accuracy for the optimization of the objectives
`repart_gene`	: (integer, length = 4) optional, number of new parameter sets for each rule and per generation
`gpp`	: (integer, length = 1) optional, calling frequency for the rule "Fireworks"
`blocks`	(optional): groups for parameters
`pop`	: (matrix, nrow = nset, ncol = nvar or nvar+nobj) optional, initial population (used to restart an optimization)
`funcinit`	(optional): the name of the initialization function applied on each node of cluster when parallel computation. The arguments are cl and numcores.
`objnames`	(optional): the name of the objectives

`listsave`	(optional): names of the listing files. Default: None (no output). If exists, fields to be defined: "pmt" (file of parameters on the Pareto Front), "obj" (file of corresponding objective values), "evol" (evolution of maximum objectives by generation). Optional field: "totalpop" (total population and corresponding objectives, useful to restart a computation)
`write_gen`	: (integer, length = 1) optional, if = 1, save files 'pmt' and 'obj' at each generation (= 0 by default)
`carallel`	: (logical, length = 1) optional, do parallel computations (TRUE by default)
`numcores`	: (integer, length = 1) optional, the number of cores for the parallel computations (all cores by default).
`graph`	: (logical, length = 1) optional, plot graphical output at each generation (TRUE by default).

Details

The optimizer was originally written for Scilab by Nicolas Le Moine. The algorithm is a hybrid of the MEAS algorithm (Efstratiadis and Koutsoyiannis (2005) <doi:10.13140/RG.2.2.32963.81446>) by using the directional search method based on the simplexes of the objective space and the epsilon-NGSA-II algorithm with the method of classification of the parameter vectors archiving management by epsilon-dominance (Reed and Devireddy <doi:10.1142/9789812567796_0004>). Documentation : "Principe de l'optimiseur CaRaMEL et illustration au travers d'exemples de parametres dans le cadre de la modelisation hydrologique conceptuelle" Frederic Hendrickx (EDF) and Nicolas Le Moine (UPMC) Report EDF H-P73-2014-09038-FR

Value

List of five elements:

success return value (logical, length = 1) : TRUE if successfull

parameters Pareto front (matrix, nrow = archsize, ncol = nvar)

objectives objectives of the Pareto front (matrix, nrow = archsize, ncol = nobj+nadditional)

save_crit evolution of the optimal objectives

total_pop total population (matrix, nrow = popsize+archsize, ncol = nvar+nobj+nadditional)

Author(s)

Fabrice Zaoui - Celine Monteil

Examples

```
# Definition of the test function
viennet <- function(i) {
  val1 <- 0.5*(x[i,1]*x[i,1]+x[i,2]*x[i,2])+sin(x[i,1]*x[i,1]+x[i,2]*x[i,2])
 val2 <- 15+(x[i,1]-x[i,2]+1)*(x[i,1]-x[i,2]+1)/27+(3*x[i,1]-2*x[i,2]+4)*(3*x[i,1]-2*x[i,2]+4)/8
  val3 <- 1/(x[i,1]*x[i,1]+x[i,2]*x[i,2]+1) -1.1*exp(-(x[i,1]*x[i,1]+x[i,2]*x[i,2]))
  return(c(val1,val2,val3))
}
# Number of objectives
```

```
nobj <- 3
# Number of variables
nvar <- 2
# All the objectives are to be minimized
minmax <- c(FALSE, FALSE, FALSE)
# Define the bound constraints
bounds <- matrix(data = 1, nrow = nvar, ncol = 2)
bounds[, 1] <- -3 * bounds[, 1]
bounds[, 2] <- 3 * bounds[, 2]

# Caramel optimization
results <-
  caRamel(nobj = nobj,
          nvar = nvar,
          minmax =  minmax,
          bounds = bounds,
          func = viennet,
          popsize = 100,
          archsize = 100,
          maxrun = 500,
          prec = matrix(1.e-3, nrow = 1, ncol = nobj),
          carallel = FALSE)
```

Cextrap — *Extrapolation along orthogonal directions to the Pareto front in the space of the objectives*

Description

gives n new candidates by extrapolation along orthogonal directions to the Pareto front in the space of the objectives

Usage

```
Cextrap(param, crit, directions, longu, n)
```

Arguments

`param`	: matrix [NPoints , NPar] of already evaluated parameters
`crit`	: matrix [Npoints , NObj] of associated criteria
`directions`	: matrix [NDir, 2] the starting and ending points of the candidate vectors
`longu`	: matrix [NDir , 1] giving the length of each segment thus defined in the OBJ space (measure of the probability of exploring this direction)
`n`	: number of new vectors to generate

Value

xnew : matrix [n , NPar] of new vectors

pcrit : matrix [n , NObj] estimated positions of new sets in the goal space

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
param <- matrix(rexp(100), 100, 1)
crit <- matrix(rexp(200), 100, 2)
directions <- matrix(c(1,3,2,7,13,40), nrow = 3, ncol = 2)
longu <- runif(3)
n <- 5
# Call the function
res <- Cextrap(param, crit, directions, longu, n)
```

Cinterp *Interpolation in simplexes of the objective space*

Description

proposes n new candidates by interpolation in simplexes of the objective space

Usage

```
Cinterp(param, crit, simplices, volume, n)
```

Arguments

`param`	: matrix [NPoints , NPar] of already evaluated parameters
`crit`	: matrix [Npoints , NObj] of associated criteria
`simplices`	: matrix [NSimp , NObj+1] containing all or part of the triangulation of the space of the objectives
`volume`	: matrix [NSimp , 1] giving the volume of each simplex (measure of the probability of interpolating in this simplex)
`n`	: number of new vectors to generate

Value

xnew : matrix [n , NPar] of new vectors

pcrit : matrix [n , NObj] estimated positions of new sets in the goal space

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
param <- matrix(rexp(100), 100, 1)
crit <- matrix(rexp(200), 100, 2)
simplices <- matrix(c(15,2,1,15,22,1,18,15,2,17,13,14), nrow = 4, ncol = 3)
volume <- runif(4)
n <- 5
# Call the function
res <- Cinterp(param, crit, simplices, volume, n)
```

`Crecombination` *Recombination of the sets of parameters*

Description

performs a recombination of the sets of parameters

Usage

```
Crecombination(param, blocks, n)
```

Arguments

`param`	: matrix [. , NPar] of the population of parameters
`blocks`	: list of integer vectors: list of variable blocks for recombination
`n`	: number of new vectors to generate

Value

xnew : matrix [n , NPar] of new vectors

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
param <- matrix(rexp(15), 15, 1)
blocks <- NULL
n <- 5
# Call the function
res <- Crecombination(param, blocks, n)
```

`Cusecovar` *New parameter vectors generation respecting a covariance structure*

Description

proposes new parameter vectors respecting a covariance structure

Usage

```
Cusecovar(xref, amplif, n)
```

Arguments

`xref`	: matrix [. , NPar] of the reference population whose covariance structure is to be used
`amplif`	: amplification factor of the standard deviation on each parameter
`n`	: number of new vectors to generate

Value

xnew : matrix [n , NPar] of new vectors

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
xref <- matrix(rexp(35), 35, 1)
amplif <- 2.
n <- 5
# Call the function
res <- Cusecovar(xref, amplif, n)
```

`decrease_pop` *Decreasing of the population of parameters sets*

Description

decreases the population of parameters sets

Usage

```
decrease_pop(matobj, minmax, prec, archsize, popsize)
```

Arguments

`matobj`	: matrix of objectives, dimension (ngames, nobj)
`minmax`	: vector of booleans, of dimension nobj: TRUE if maximization of the objective, FALSE otherwise
`prec`	: nobj dimension vector: accuracy
`archsize`	: integer: archive size
`popsize`	: integer: population size

Value

A list containing two elements:

ind_arch indices of individuals in the updated Pareto front

ind_pop indices of individuals in the updated population

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
matobj <- matrix(rexp(200), 100, 2)
prec <- c(1.e-3, 1.e-3)
archsize <- 100
minmax <- c(FALSE, FALSE)
popsize <- 100
# Call the function
res <- decrease_pop(matobj, minmax, prec, archsize, popsize)
```

`Dimprove` *Determination of directions for improvement*

Description

determines directions for improvement

Usage

```
Dimprove(o_splx, f_splx)
```

Arguments

`o_splx`	: matrix of objectives of simplexes (nrow = npoints, ncol = nobj)
`f_splx`	: vector (npoints) of associated Pareto numbers (1 = dominated)

Value

list of elements "oriedge": oriented edges and "ledge": length

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
o_splx <- matrix(rexp(6), 3, 2)
f_splx <- c(1,1,1)
# Call the function
res <- Dimprove(o_splx, f_splx)
```

dominate *Successive Pareto fronts of a population*

Description

calculates the successive Pareto fronts of a population (classification "onion peel"), when objectives need to be maximized.

Usage

```
dominate(matobj)
```

Arguments

`matobj`	: matrix [NInd , NObj] of objectives

Value

f : vector of dimension NInd of dominances

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
matobj <- matrix(runif(200), 100, 2)
# Call the function
pareto_rank <- dominate(matobj)
```

dominated *Rows domination of a matrix by a vector*

Description

indicates which rows of the matrix Y are dominated by the vector (row) x

Usage

```
dominated(x, Y)
```

Arguments

`x`	: row vecteur
`Y`	: matrix

Value

D : vector of booleans

Author(s)

F. Zaoui

Examples

```
# Definition of the parameters
Y <- matrix(rexp(200), 100, 2)
x <- Y[1,]
# Call the function
res <- dominated(x, Y)
```

downsize *Downsizing of a population to only one individual per box up to a given accuracy*

Description

reduces the number of individuals in a population to only one individual per box up to a given accuracy

Usage

```
downsize(points, Fo, prec)
```

Arguments

`points`	: matrix of objectives
`Fo`	: rank on the front of each point (1: dominates on the Pareto)
`prec`	: (double, length = nobj) desired accuracy for sorting objectives

Value

vector indices

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
points <- matrix(rexp(200), 100, 2)
prec <- c(1.e-3, 1.e-3)
Fo <- sample(1:100, 100)
# Call the function
res <- downsize(points, Fo, prec)
```

`matvcov` *Calculation of the variances-covariances matrix on the reference population*

Description

calculates the variances-covariances matrix on the reference population

Usage

```
matvcov(x, g)
```

Arguments

`x`	: population
`g`	: center of reference population (in the parameter space)

Value

rr : variances-covariances matrix on the reference population

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
x <- matrix(rexp(30), 30, 1)
g <- mean(x)
# Call the function
res <- matvcov(x, g)
```

`newXval` *Generation of a new population of parameter sets following the five rules of caRamel*

Description

generates a new population of parameter sets following the five rules of caRamel

Usage

```
newXval(param, crit, isperf, sp, bounds, repart_gene, blocks, fireworks)
```

Arguments

`param`	: matrix [Nvec , NPar] of parameters of the current population
`crit`	: matrix [Nvec , NObj] of associated criteria
`isperf`	: vector of Booleans of length NObj, TRUE if maximization of the objective, FALSE otherwise
`sp`	: variance a priori of the parameters
`bounds`	: lower and upper bounds of parameters [NPar , 2]
`repart_gene`	: matrix of length 4 giving the number of games to be generated with each rule: 1 Interpolation in the simplexes of the front, 2 Extrapolation according to the directions of the edges "orthogonal" to the front, 3 Random draws with prescribed variance-covariance matrix, 4 Recombination by functional blocks
`blocks`	: list of integer vectors containing function blocks of parameters
`fireworks`	: boolean, TRUE if one tests a random variation on each parameter and each maximum of O.F.

Value

xnew : matrix of new vectors [sum(Repart_Gene) + eventually (nobj+1)*nvar if fireworks , NPar]

project_crit: assumed position of the new vectors in the criteria space: [sum(Repart_Gene)+ eventually (nobj+1)*nvar if fireworks , NObj];

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
param <- matrix(rexp(100), 100, 1)
crit <- matrix(rexp(200), 100, 2)
isperf <- c(FALSE, FALSE)
bounds <- matrix(data = 1, nrow = 1, ncol = 2)
bounds[, 1] <- -5 * bounds[, 1]
bounds[, 2] <- 10 * bounds[, 2]
sp <- (bounds[, 2] - bounds[, 1]) / (2 * sqrt(3))
repart_gene <- c(5, 5, 5, 5)
fireworks <- TRUE
blocks <- NULL
# Call the function
res <- newXval(param, crit, isperf, sp, bounds, repart_gene, blocks, fireworks)
```

`pareto` *Indicates which rows are Pareto*

Description

indicates which rows of the X criterion matrix are Pareto, when objectives need to be maximized

Usage

```
pareto(X)
```

Arguments

`X`	: matrix of objectives [NInd * NObj]

Value

Ft : vector [NInd], TRUE when the set is on the Pareto front.

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
X <- matrix(runif(200), 100, 2)
# Call the function
is_pareto <- pareto(X)
```

plot_caramel *Plotting of caRamel results*

Description

Plot graphs of the Pareto front and a graph of optimization evolution

Usage

```
plot_caramel(caramel_results, nobj = NULL, objnames = NULL)
```

Arguments

`caramel_results`	: list resulting from the caRamel() function, with fields $objectives and $save_crit
`nobj`	: number of objectives (optional)
`objnames`	: vector of objectives names (optional)

Examples

```
# Definition of the test function
viennet <- function(i) {
  val1 <- 0.5*(x[i,1]*x[i,1]+x[i,2]*x[i,2])+sin(x[i,1]*x[i,1]+x[i,2]*x[i,2])
 val2 <- 15+(x[i,1]-x[i,2]+1)*(x[i,1]-x[i,2]+1)/27+(3*x[i,1]-2*x[i,2]+4)*(3*x[i,1]-2*x[i,2]+4)/8
  val3 <- 1/(x[i,1]*x[i,1]+x[i,2]*x[i,2]+1) -1.1*exp(-(x[i,1]*x[i,1]+x[i,2]*x[i,2]))
  return(c(val1,val2,val3))
}
nobj <- 3 # Number of objectives
nvar <- 2 # Number of variables
minmax <- c(FALSE, FALSE, FALSE) # All the objectives are to be minimized
bounds <- matrix(data = 1, nrow = nvar, ncol = 2) # Define the bound constraints
bounds[, 1] <- -3 * bounds[, 1]
bounds[, 2] <- 3 * bounds[, 2]

# Caramel optimization
results <- caRamel(nobj, nvar, minmax, bounds, viennet, popsize = 100, archsize = 100,
         maxrun = 500, prec = matrix(1.e-3, nrow = 1, ncol = nobj), carallel = FALSE)

# Plot of results
plot_caramel(results)
```

`plot_pareto` *Plotting of a population of objectives and Pareto front*

Description

Plots graphs the population regarding each couple of objectives and emphasizes the Pareto front

Usage

```
plot_pareto(MatObj, nobj = NULL, objnames = NULL, maximized = NULL)
```

Arguments

`MatObj`	: matrix of the objectives [NInd, nobj]
`nobj`	: number of objectives (optional)
`objnames`	: vector, length nobj, of names of the objectives (optional)
`maximized`	: vector of logical, length nobj, TRUE if objective need to be maximized, FALSE if minimized

Author(s)

C. Monteil

Examples

```
# Definition of the population
Pop <- matrix(runif(300), 100, 3)

# Definition of objectives to maximize (Obj1, Obj2) and to minimize (Obj3)
maximized <- c(TRUE,TRUE,FALSE)

# Call the function
plot_pareto(MatObj = Pop, maximized=maximized)
```

`plot_population` *Plotting of a population of objectives*

Description

Plot graphs the population regarding each couple of objectives

Usage

```
plot_population(MatObj, nobj, ngen = NULL, nrun = NULL, objnames = NULL,
  MatEvol = NULL, popsize = 0)
```

Arguments

`MatObj`	: matrix of the objectives [NInd, nobj]
`nobj`	: number of objectives
`ngen`	: number of generations (optional)
`nrun`	: number of model evaluations (optional)
`objnames`	: vector of objectives names (optional)
`MatEvol`	: matrix of the evolution of the optimal objectives (optional)
`popsize`	: integer, size of the initial population (optional)

Author(s)

C. Monteil

Examples

```
# Definition of the population
Pop <- matrix(runif(300), 100, 3)
# Call the function
plot_population(MatObj = Pop, nobj=3, objnames=c("Obj1","Obj2","Obj3"))
```

`rselect` *Selection of n points*

Description

performs a selection of n points in facp

Usage

```
rselect(n, facp)
```

Arguments

`n`	: number of points to select
`facp`	: vector of initial points

Value

ix : ranks of selected points (vector of dimension n)

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
n <- 5
facp <- runif(30)
# Call the function
res <- rselect(n, facp)
```

`val2rank` *Converting the values of a vector into their rank*

Description

converts the values of a vector into their rank

Usage

```
val2rank(X, opt)
```

Arguments

`X`	: vector to treat
`opt`	: integer which gives the rule to follow in case of tied ranks (repeated values): if opt = 1, one returns the average rank, if opt = 2, one returns the corresponding rank in the series of the unique values, if opt = 3, return the max rank

Value

R : rank vector

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
X <- matrix(rexp(100), 100, 1)
opt <- 3
# Call the function
res <- val2rank(X, opt)
```

vol_splx *Volume of a simplex*

Description

calculates the volume of a simplex

Usage

```
vol_splx(S)
```

Arguments

S	: matrix (d+1) rows * d columns containing the coordinates in d-dim of d + 1 vertices of a simplex

Value

V : simplex volume

Author(s)

Fabrice Zaoui

Examples

```
# Definition of the parameters
S <- matrix(rexp(6), 3, 2)
# Call the function
res <- vol_splx(S)
```

Index