Package ‘DiceView’

January 20, 2025

Title Methods for Visualization of Computer Experiments Design and Surrogate

Version 3.1-0

Date 2024-11-12

Maintainer Yann Richet <yann.richet@irsn.fr>

Description View 2D/3D sections, contour plots, mesh of excursion sets for computer experiments designs, surrogates or test functions.

Depends methods, utils, stats, grDevices, graphics

Imports DiceDesign, R.cache, geometry, scatterplot3d, parallel, foreach

Suggests rlibkriging, DiceKriging, DiceEval, rgl, arrangements

License GPL-3

URL https://github.com/IRSN/DiceView

Repository CRAN

RoxygenNote 7.3.1

Encoding UTF-8

NeedsCompilation no

Author Yann Richet [aut, cre] (<https://orcid.org/0000-0002-5677-8458>), Yves Deville [aut], Clement Chevalier [ctb]

Date/Publication 2024-11-12 17:40:02 UTC

Contents

`Apply.function` *Apply Functions Over Array Margins, using custom vectorization (possibly using parallel)*

Description

Emulate parallel apply on a function, from mclapply. Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

Usage

```
Apply.function(
  FUN,
  X,
  MARGIN = 1,
  .combine = c,
  .lapply = parallel::mclapply,
  ...
)
```

Arguments

`FUN`	function to apply on X
`X`	array of input values for FUN
`MARGIN`	1 indicates to apply on rows (default), 2 on columns
`.combine`	how to combine results (default using c(.))
`.lapply`	how to vectorize FUN call (default is parallel::mclapply)
`...`	optional arguments to FUN.

Value

array of values taken by FUN on each row/column of X

Examples

```
X = matrix(runif(10),ncol=2);
  rowSums(X) == apply(X,1,sum)
  apply(X,1,sum) == Apply.function(sum,X)

X = matrix(runif(10),ncol=1)
  rowSums(X) == apply(X,1,sum)
  apply(X,1,sum) == Apply.function(sum,X)

X = matrix(runif(10),ncol=2)
f = function(X) X[1]/X[2]
apply(X,1,f) == Apply.function(f,X)
```

`are_in.mesh` *Checks if some points belong to a given mesh*

Description

Checks if some points belong to a given mesh

Usage

```
are_in.mesh(X, mesh)
```

Arguments

`X`	points to check
`mesh`	mesh identifying the set which X may belong

Examples

```
X = matrix(runif(100),ncol=2);
inside = are_in.mesh(X,mesh=geometry::delaunayn(matrix(c(0,0,1,1,0,0),ncol=2),output.options =TRUE))
print(inside)
plot(X,col=rgb(1-inside,0,0+inside))
```

`branin` *This is a simple copy of the Branin-Hoo 2-dimensional test function, as provided in DiceKriging package. The Branin-Hoo function is defined here over [0,1] x [0,1], instead of [-5,0] x [10,15] as usual. It has 3 global minima : x1 = c(0.9616520, 0.15); x2 = c(0.1238946, 0.8166644); x3 = c(0.5427730, 0.15)*

Description

This is a simple copy of the Branin-Hoo 2-dimensional test function, as provided in DiceKriging package. The Branin-Hoo function is defined here over [0,1] x [0,1], instead of [-5,0] x [10,15] as usual. It has 3 global minima : x1 = c(0.9616520, 0.15); x2 = c(0.1238946, 0.8166644); x3 = c(0.5427730, 0.15)

Usage

```
branin(x)
```

Arguments

`x`	a 2-dimensional vector specifying the location where the function is to be evaluated.

Value

A real number equal to the Branin-Hoo function values at x

`combn.design` *Generalize expand.grid() for multi-columns data. Build all combinations of lines from X1 and X2. Each line may hold multiple columns.*

Description

Generalize expand.grid() for multi-columns data. Build all combinations of lines from X1 and X2. Each line may hold multiple columns.

Usage

```
combn.design(X1, X2)
```

Arguments

`X1`	variable values, possibly with many columns
`X2`	variable values, possibly with many columns combn.design(matrix(c(10,20),ncol=1),matrix(c(1,2,3,4,5,6) combn.design(matrix(c(10,20,30,40),ncol=2),matrix(c(1,2,3,4,5,6),ncol=2))

contourview.function *Plot a contour view of a prediction model or function, including design points if available.*

Description

Plot a contour view of a prediction model or function, including design points if available.

Usage

```
## S3 method for class '`function`'
contourview(
  fun,
  vectorized = FALSE,
  dim = NULL,
  center = NULL,
  lty_center = 2,
  col_center = "black",
  axis = NULL,
  npoints = 21,
  levels = 10,
  lty_levels = 3,
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels - 1) else
    col.levels("blue", levels - 1),
  col = NULL,
  col_fading_interval = 0.5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'matrix'
contourview(
  X,
  y,
  center = NULL,
  lty_center = 2,
  col_center = "black",
  axis = NULL,
  col_points = if (!is.null(col)) col else "red",
  col = NULL,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'character'
contourview(eval_str, axis = NULL, mfrow = NULL, ...)

## S3 method for class 'km'
contourview(
  km_model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(km_model@y, 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
  col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'Kriging'
contourview(
  Kriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(Kriging_model$y(), 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
  col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'NuggetKriging'
contourview(
  NuggetKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(NuggetKriging_model$y(), 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
  col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'NoiseKriging'
contourview(
  NoiseKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(NoiseKriging_model$y(), 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
  col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'glm'
contourview(
  glm_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(glm_model$fitted.values, 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
   col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'list'
contourview(
  modelFit_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(modelFit_model$data$Y, 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
   col.levels("blue", levels),
  col = NULL,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

contourview(...)
```

Arguments

`fun`	a function or 'predict()'-like function that returns a simple numeric or mean and standard error: list(mean=...,se=...).
`vectorized`	is fun vectorized?
`dim`	input variables dimension of the model or function.
`center`	optional coordinates (as a list or data frame) of the center of the section view if the model's dimension is > 2.
`lty_center`	line type for the section center of the plot (if any).
`col_center`	color for the section center of the plot (if any).
`axis`	optional matrix of 2-axis combinations to plot, one by row. The value `NULL` leads to all possible combinations i.e. `choose(D, 2)`.
`npoints`	an optional number of points to discretize plot of response surface and uncertainties.
`levels`	(number of) contour levels to display.
`lty_levels`	contour line type.
`col_levels`	color for the surface.
`col`	color of the object (use col_* for specific objects).
`col_fading_interval`	an optional factor of alpha (color channel) fading used to plot function output intervals (if any).
`mfrow`	an optional list to force `par(mfrow = ...)` call. The default value `NULL` is automatically set for compact view.
`Xlab`	an optional list of string to overload names for X.
`ylab`	an optional string to overload name for y.
`Xlim`	an optional list to force x range for all plots. The default value `NULL` is automatically set to include all design points.
`title`	an optional overload of main title.
`add`	to print graphics on an existing window.
`...`	arguments of the `contourview.km`, `contourview.glm`, `contourview.Kriging` or `contourview.function` function
`X`	the matrix of input design.
`y`	the array of output values (two columns means an interval).
`col_points`	color of points.
`bg_fading`	an optional factor of alpha (color channel) fading used to plot design points outside from this section.

`eval_str`	the expression to evaluate in each subplot.
`km_model`	an object of class `"km"`.
`type`	the kriging type to use for model prediction.
`conf_level`	confidence hulls to display.
`conf_fading`	an optional factor of alpha (color channel) fading used to plot confidence hull.
`Kriging_model`	an object of class `"Kriging"`.
`NuggetKriging_model`	an object of class `"Kriging"`.
`NoiseKriging_model`	an object of class `"Kriging"`.
`glm_model`	an object of class `"glm"`.
`modelFit_model`	an object returned by DiceEval::modelFit.

Details

If available, experimental points are plotted with fading colors. Points that fall in the specified section (if any) have the color specified `col_points` while points far away from the center have shaded versions of the same color. The amount of fading is determined using the Euclidean distance between the plotted point and `center`.

Author(s)

Yann Richet, IRSN

See Also

`sectionview.function` for a section plot, and `sectionview3d.function` for a 2D section plot.

`sectionview.matrix` for a section plot, and `sectionview3d.matrix` for a 2D section plot.

`contourview.matrix` for a section plot.

`sectionview.km` for a section plot, and `sectionview3d.km` for a 2D section plot.

`sectionview.Kriging` for a section plot, and `sectionview3d.Kriging` for a 2D section plot.

`sectionview.NuggetKriging` for a section plot, and `sectionview3d.NuggetKriging` for a 2D section plot.

`sectionview.NoiseKriging` for a section plot, and `sectionview3d.NoiseKriging` for a 2D section plot.

`sectionview.glm` for a section plot, and `sectionview3d.glm` for a 2D section plot.

`sectionview.glm` for a section plot, and `sectionview3d.glm` for a 2D section plot.

Examples

```
x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2 + rnorm(15)
model <- lm(y ~ x1 + x2)
```

```
contourview(function(x) sum(x),
                     dim=2, Xlim=cbind(range(x1),range(x2)), col='black')
points(x1,x2)

contourview(function(x) {
                      x = as.data.frame(x)
                      colnames(x) <- all.vars(model$call)[-1]
                      predict.lm(model, newdata=x, se.fit=FALSE)
                    }, vectorized=TRUE, dim=2,
              Xlim=cbind(range(x1),range(x2)), add=TRUE)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

contourview(X, y)

x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2^2 + rnorm(15)
model <- glm(y ~ x1 + I(x2^2))

contourview(model)

contourview("abline(h=0.25,col='red')")
if (requireNamespace("DiceKriging")) { library(DiceKriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- km(design = X, response = y, covtype="matern3_2")

contourview(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- Kriging(X = X, y = y, kernel="matern3_2")

contourview(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)
model <- NuggetKriging(X = X, y = y, kernel="matern3_2")

contourview(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)

model <- NoiseKriging(X = X, y = y, kernel="matern3_2", noise=rep(5^2,15))

contourview(model)

}

x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2^2 + rnorm(15)
model <- glm(y ~ x1 + I(x2^2))

contourview(model)

if (requireNamespace("DiceEval")) { library(DiceEval)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- modelFit(X, y, type = "StepLinear")

contourview(model)

}

## A 2D example - Branin-Hoo function
contourview(branin, dim=2, levels=30, col='black')

## Not run:
## a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
design.fact <- expand.grid(seq(0, 1, length = 4), seq(0, 1, length = 4))
design.fact <- data.frame(design.fact); names(design.fact) <- c("x1", "x2")
y <- branin(design.fact); names(y) <- "y"

if (requireNamespace("DiceKriging")) { library(DiceKriging)
## model: km
model <- DiceKriging::km(design = design.fact, response = y)
contourview(model, levels=30)
contourview(branin, dim=2, levels=30, col='red', add=TRUE)
}
```

```
if (requireNamespace("rlibkriging")) { library(rlibkriging)
## model: Kriging
model <- Kriging(X = as.matrix(design.fact), y = as.matrix(y), kernel="matern3_2")
contourview(model, levels=30)
contourview(branin, dim=2, levels=30, col='red', add=TRUE)
}

## model: glm
model <- glm(y ~ 1+ x1 + x2 + I(x1^2) + I(x2^2) + x1*x2, data=cbind(y,design.fact))
contourview(model, levels=30)
contourview(branin, dim=2, levels=30, col='red', add=TRUE)

if (requireNamespace("DiceEval")) { library(DiceEval)
## model: StepLinear
model <- modelFit(design.fact, y, type = "StepLinear")
contourview(model, levels=30)
contourview(branin, dim=2, levels=30, col='red', add=TRUE)
}

## End(Not run)
```

`EvalInterval.function` *eval function and cast result to a list of y, y_low, y_up (possibly NA)*

Description

eval function and cast result to a list of y, y_low, y_up (possibly NA)

Usage

```
EvalInterval.function(fun, X, vectorized = FALSE, dim = ncol(X))
```

Arguments

`fun`	function to evaluate
`X`	matrix of input values for fun
`vectorized`	whether fun is vectorized or not
`dim`	dimension of input values for fun if

Value

list of y, y_low, y_up

expand.grids — *Create a Data Frame from all combinations of factor variables*

Description

Generalization of base::expand.grid to more than 2 variables.

Usage

```
expand.grids(d = length(list(...)), ...)
```

Arguments

`d`	number of variables (taken in following arguments with modulo)
`...`	variables to combine, as arrays of values

Value

data frame of all possible combinations of variables values

filledcontourview.function — *Plot a contour view of a prediction model or function, including design points if available.*

Description

Plot a contour view of a prediction model or function, including design points if available.

Usage

```
## S3 method for class '`function`'
filledcontourview(
  fun,
  vectorized = FALSE,
  dim = NULL,
  center = NULL,
  lty_center = 2,
  col_center = "black",
  axis = NULL,
  npoints = 21,
  levels = 10,
  lty_levels = 1,
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels - 1) else
    col.levels("blue", levels - 1),
  col = NULL,
  col_fading_interval = 0.5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'km'
filledcontourview(
  km_model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(km_model@y, 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'Kriging'
filledcontourview(
  Kriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(Kriging_model$y(), 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'NuggetKriging'
filledcontourview(
  NuggetKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(NuggetKriging_model$y(), 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'NoiseKriging'
filledcontourview(
  NoiseKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(NoiseKriging_model$y(), 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'glm'
filledcontourview(
  glm_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(glm_model$fitted.values, 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  conf_level = 0.5,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'list'
filledcontourview(
  modelFit_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  levels = pretty(modelFit_model$data$Y, 10),
  col_points = if (!is.null(col) & length(col) == 1) col else "red",
 col_levels = if (!is.null(col) & length(col) == 1) col.levels(col, levels) else
    col.levels("blue", levels),
  col = NULL,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

filledcontourview(...)
```

Arguments

`fun`	a function or 'predict()'-like function that returns a simple numeric or mean and standard error: list(mean=...,se=...).
`vectorized`	is fun vectorized?
`dim`	input variables dimension of the model or function.
`center`	optional coordinates (as a list or data frame) of the center of the section view if the model's dimension is > 2.
`lty_center`	line type for thesection center of the plot (if any).
`col_center`	color for the section center of the plot (if any).
`axis`	optional matrix of 2-axis combinations to plot, one by row. The value `NULL` leads to all possible combinations i.e. `choose(D, 2)`.
`npoints`	an optional number of points to discretize plot of response surface and uncertainties.
`levels`	(number of) contour levels to display.
`lty_levels`	contour line type.
`col_levels`	color for the surface.
`col`	color of the object (use col_* for specific objects).
`col_fading_interval`	an optional factor of alpha (color channel) fading used to plot function output intervals (if any).
`mfrow`	an optional list to force `par(mfrow = ...)` call. The default value `NULL` is automatically set for compact view.
`Xlab`	an optional list of string to overload names for X.
`ylab`	an optional string to overload name for y.
`Xlim`	an optional list to force x range for all plots. The default value `NULL` is automatically set to include all design points.
`title`	an optional overload of main title.
`add`	to print graphics on an existing window.
`...`	arguments of the `filledcontourview.km`, `filledcontourview.glm`, `filledcontourview.Kriging` or `filledcontourview.function` function
`km_model`	an object of class `"km"`.
`type`	the kriging type to use for model prediction.
`col_points`	color of points.
`conf_level`	confidence hulls to display.

`conf_fading`	an optional factor of alpha (color channel) fading used to plot confidence hull.
`bg_fading`	an optional factor of alpha (color channel) fading used to plot design points outside from this section.
`Kriging_model`	an object of class `"Kriging"`.
`NuggetKriging_model`	an object of class `"Kriging"`.
`NoiseKriging_model`	an object of class `"Kriging"`.
`glm_model`	an object of class `"glm"`.
`modelFit_model`	an object returned by DiceEval::modelFit.

Details

If available, experimental points are plotted with fading colors. Points that fall in the specified section (if any) have the color specified `col_points` while points far away from the center have shaded versions of the same color. The amount of fading is determined using the Euclidean distance between the plotted point and `center`.

Author(s)

Yann Richet, IRSN

See Also

`sectionview.function` for a section plot, and `sectionview3d.function` for a 2D section plot.

`sectionview.km` for a section plot, and `sectionview3d.km` for a 2D section plot.

`sectionview.Kriging` for a section plot, and `sectionview3d.Kriging` for a 2D section plot.

`sectionview.NuggetKriging` for a section plot, and `sectionview3d.NuggetKriging` for a 2D section plot.

`sectionview.NoiseKriging` for a section plot, and `sectionview3d.NoiseKriging` for a 2D section plot.

`sectionview.glm` for a section plot, and `sectionview3d.glm` for a 2D section plot.

`sectionview.glm` for a section plot, and `sectionview3d.glm` for a 2D section plot.

Examples

```
x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2 + rnorm(15)
model <- lm(y ~ x1 + x2)

filledcontourview(function(x) sum(x),
                     dim=2, Xlim=cbind(range(x1),range(x2)), col='black')
points(x1,x2)

filledcontourview(function(x) {
```

```
                      x = as.data.frame(x)
                      colnames(x) <- all.vars(model$call)[-1]
                      predict.lm(model, newdata=x, se.fit=FALSE)
                    }, vectorized=TRUE, dim=2,
                  Xlim=cbind(range(x1),range(x2)), add=TRUE)

if (requireNamespace("DiceKriging")) { library(DiceKriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- km(design = X, response = y, covtype="matern3_2")

filledcontourview(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- Kriging(X = X, y = y, kernel="matern3_2")

filledcontourview(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)

model <- NuggetKriging(X = X, y = y, kernel="matern3_2")

filledcontourview(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)

model <- NoiseKriging(X = X, y = y, kernel="matern3_2", noise=rep(5^2,15))

filledcontourview(model)

}

x1 <- rnorm(15)
x2 <- rnorm(15)
```

```
y <- x1 + x2^2 + rnorm(15)
model <- glm(y ~ x1 + I(x2^2))

filledcontourview(model)

if (requireNamespace("DiceEval")) { library(DiceEval)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- modelFit(X, y, type = "StepLinear")

filledcontourview(model)

}

## A 2D example - Branin-Hoo function
filledcontourview(branin, dim=2, levels=30, col='black')

## Not run:
## a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
design.fact <- expand.grid(seq(0, 1, length = 4), seq(0, 1, length = 4))
design.fact <- data.frame(design.fact); names(design.fact) <- c("x1", "x2")
y <- branin(design.fact); names(y) <- "y"

if (requireNamespace("DiceKriging")) { library(DiceKriging)
## model: km
model <- DiceKriging::km(design = design.fact, response = y)
filledcontourview(model, levels=30)
filledcontourview(branin, dim=2, levels=30, col='red', add=TRUE)
}

if (requireNamespace("rlibkriging")) { library(rlibkriging)
## model: Kriging
model <- Kriging(X = as.matrix(design.fact), y = as.matrix(y), kernel="matern3_2")
filledcontourview(model, levels=30)
filledcontourview(branin, dim=2, levels=30, col='red', add=TRUE)
}

## model: glm
model <- glm(y ~ 1+ x1 + x2 + I(x1^2) + I(x2^2) + x1*x2, data=cbind(y,design.fact))
filledcontourview(model, levels=30)
filledcontourview(branin, dim=2, levels=30, col='red', add=TRUE)

if (requireNamespace("DiceEval")) { library(DiceEval)
## model: StepLinear
model <- modelFit(design.fact, y, type = "StepLinear")
filledcontourview(model, levels=30)
filledcontourview(branin, dim=2, levels=30, col='red', add=TRUE)
}

## End(Not run)
```

`is.mesh` *Checks if a mesh is valid*

Description

Checks if a mesh is valid

Usage

```
is.mesh(x)
```

Arguments

`x`	mesh to check

Value

TRUE if mesh is valid

`is_in.mesh` *Checks if some point belongs to a given mesh*

Description

Checks if some point belongs to a given mesh

Usage

```
is_in.mesh(x, mesh)
```

Arguments

`x`	point to check
`mesh`	mesh identifying the set which X may belong

Examples

```
is_in.mesh(-0.5,mesh=geometry::delaunayn(matrix(c(0,1),ncol=1),output.options =TRUE))
is_in.mesh(0.5,mesh=geometry::delaunayn(matrix(c(0,1),ncol=1),output.options =TRUE))

x =matrix(-.5,ncol=2,nrow=1)
is_in.mesh(x,mesh=geometry::delaunayn(matrix(c(0,0,1,1,0,0),ncol=2),output.options =TRUE))

x =matrix(.5,ncol=2,nrow=1)
is_in.mesh(x,mesh=geometry::delaunayn(matrix(c(0,0,1,1,0,0),ncol=2),output.options =TRUE))
```

`is_in.p` *Test if points are in a hull*

Description

Test if points are in a hull

Usage

```
is_in.p(x, p, h = NULL)
```

Arguments

`x`	points to test
`p`	points defining the hull
`h`	hull itself (built from p if given as NULL (default))

Examples

```
is_in.p(x=-0.5,p=matrix(c(0,1),ncol=1))
is_in.p(x=0.5,p=matrix(c(0,1),ncol=1))
is_in.p(x=matrix(-.5,ncol=2,nrow=1),p=matrix(c(0,0,1,1,0,0),ncol=2))
is_in.p(x=matrix(.25,ncol=2,nrow=1),p=matrix(c(0,0,1,1,0,0),ncol=2))
is_in.p(x=matrix(-.5,ncol=3,nrow=1),p=matrix(c(0,0,0,1,0,0,0,1,0,0,0,1),ncol=3,byrow = TRUE))
is_in.p(x=matrix(.25,ncol=3,nrow=1),p=matrix(c(0,0,0,1,0,0,0,1,0,0,0,1),ncol=3,byrow = TRUE))
```

`Memoize.function` *Memoize a function*

Description

Before each call of a function, check that the cache holds the results and returns it if available. Otherwise, compute f and cache the result for next evluations.

Usage

```
Memoize.function(fun)
```

Arguments

`fun`	function to memoize

Value

a function with same behavior than argument one, but using cache.

Examples

```
f=function(n) rnorm(n);
F=Memoize.function(f);
F(5); F(6); F(5)
```

---

mesh *Builds a mesh from a design aor set of points*

Description

Builds a mesh from a design aor set of points

Usage

```
mesh(intervals, mesh.type = "seq", mesh.sizes = 11)
```

Arguments

`intervals`	bounds to inverse in, each column contains min and max of each dimension
`mesh.type`	function or "unif" or "seq" (default) or "LHS" to preform interval partition
`mesh.sizes`	number of parts for mesh (duplicate for each dimension if using "seq")

Value

delaunay mesh (list(p,tri,...) from geometry)

Examples

```
mesh = mesh(intervals=matrix(c(0,1,0,1),ncol=2),mesh.type="unif",mesh.sizes=10)
plot2d_mesh(mesh)
```

---

mesh_exsets *Search excursion set of nD function, sampled by a mesh*

Description

Search excursion set of nD function, sampled by a mesh

Usage

```
mesh_exsets(
  f,
  vectorized = FALSE,
  threshold,
  sign,
  intervals,
  mesh.type = "seq",
  mesh.sizes = 11,
  maxerror_f = 1e-09,
  tol = .Machine$double.eps^0.25,
  ex_filter.tri = all,
  ...
)
```

Arguments

`f`	Function to inverse at 'threshold'
`vectorized`	boolean: is f already vectorized ? (default: FALSE) or if function: vectorized version of f.
`threshold`	target value to inverse
`sign`	focus at conservative for above (sign=1) or below (sign=-1) the threshold
`intervals`	bounds to inverse in, each column contains min and max of each dimension
`mesh.type`	"unif" or "seq" (default) or "LHS" to preform interval partition
`mesh.sizes`	number of parts for mesh (duplicate for each dimension if using "seq")
`maxerror_f`	maximal tolerance on f precision
`tol`	the desired accuracy (convergence tolerance on f arg).
`ex_filter.tri`	boolean function to validate a geometry::tri as considered in excursion : 'any' or 'all'
`...`	parameters to forward to roots_mesh(...) call

Examples

```
# mesh_exsets(function(x) x, threshold=.51, sign=1, intervals=rbind(0,1),
#   maxerror_f=1E-2,tol=1E-2) # for faster testing
# mesh_exsets(function(x) x, threshold=.50000001, sign=1, intervals=rbind(0,1),
#   maxerror_f=1E-2,tol=1E-2) # for faster testing
# mesh_exsets(function(x) sum(x), threshold=.51,sign=1, intervals=cbind(rbind(0,1),rbind(0,1)),
#   maxerror_f=1E-2,tol=1E-2) # for faster testing
# mesh_exsets(sin,threshold=0,sign="sup",interval=c(pi/2,5*pi/2),
#   maxerror_f=1E-2,tol=1E-2) # for faster testing

if (identical(Sys.getenv("NOT_CRAN"), "true")) { # too long for CRAN on Windows

  e = mesh_exsets(function(x) (0.25+x[1])^2+(0.5+x[2])^2 ,
              threshold =0.25,sign=-1, intervals=matrix(c(-1,1,-1,1),nrow=2),
                maxerror_f=1E-2,tol=1E-2) # for faster testing

  plot(e$p,xlim=c(-1,1),ylim=c(-1,1));
  apply(e$tri,1,function(tri) polygon(e$p[tri,],col=rgb(.4,.4,.4,.4)))

  if (requireNamespace("rgl")) {
    e = mesh_exsets(function(x) (0.5+x[1])^2+(-0.5+x[2])^2+(0.+x[3])^2,
                threshold = .25,sign=-1, mesh.type="unif",
                intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2),
                maxerror_f=1E-2,tol=1E-2) # for faster testing

    rgl::plot3d(e$p,xlim=c(-1,1),ylim=c(-1,1),zlim=c(-1,1));
    apply(e$tri,1,function(tri)rgl::lines3d(e$p[tri,]))
  }
}
```

mesh_level *Mesh level set of function*

Description

Mesh level set of function

Usage

```
mesh_level(f, vectorized = FALSE, level = 0, intervals, mesh, ...)
```

Arguments

`f`	function to be evaluated on the mesh
`vectorized`	logical or function. If TRUE, f is assumed to be vectorized.
`level`	level/threshold value
`intervals`	matrix of intervals
`mesh`	mesh object or type
`...`	additional arguments passed to f

`min_dist` *Minimal distance between one point to many points*

Description

Minimal distance between one point to many points

Usage

```
min_dist(x, X, norm = rep(1, ncol(X)))
```

Arguments

`x`	one point
`X`	matrix of points (same number of columns than x)
`norm`	normalization vecor of distance (same number of columns than x)

Value

minimal distance

Examples

```
min_dist(runif(3),matrix(runif(30),ncol=3))
```

`optims` *Title Multi-local optimization wrapper for optim, using (possibly parallel) multistart.*

Description

Title Multi-local optimization wrapper for optim, using (possibly parallel) multistart.

Usage

```
optims(
  pars,
  fn,
  fn.NaN = NaN,
  .apply = "mclapply",
  pars.eps = 1e-05,
  control = list(),
  ...
)
```

Arguments

`pars`	starting points for optim
`fn`	objective function, like in optim().
`fn.NaN`	replacement value of fn when returns NaN
`.apply`	loop/parallelization backend for multistart ("mclapply", "lapply" or "foreach")
`pars.eps`	minimal distance between two solutions to be considered different
`control`	control parameters for optim()
`...`	additional arguments passed to optim()

Value

list with best solution and all solutions

Author(s)

Yann Richet, IRSN

Examples

```
fn = function(x) ifelse(x==0,1,sin(x)/x)
# plot(fn, xlim=c(-20,20))
optim( par=5,                    fn,lower=-20,upper=20,method='L-BFGS-B')
optims(pars=t(t(seq(-20,20,,20))),fn,lower=-20,upper=20,method='L-BFGS-B')

# Branin function (3 local minimas)
f = function (x) {
  x1 <- x[1] * 15 - 5
  x2 <- x[2] * 15
  (x2 - 5/(4 * pi^2) * (x1^2) + 5/pi * x1 - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(x1) + 10
}
# expect to find 3 local minimas
optims(pars=matrix(runif(100),ncol=2),f,method="L-BFGS-B",lower=c(0,0),upper=c(1,1))
```

`plot2d_mesh` *Plot a two dimensional mesh*

Description

Plot a two dimensional mesh

Usage

```
plot2d_mesh(
  mesh,
  color.nodes = "black",
  color.mesh = "darkgray",
  alpha = 0.4,
  ...
)
```

Arguments

`mesh`	2-dimensional mesh to draw
`color.nodes`	color of the mesh nodes
`color.mesh`	color of the mesh elements
`alpha`	transparency of the mesh elements & nodes
`...`	optional arguments passed to plot function

Examples

```
plot2d_mesh(mesh_exsets(f = function(x) sin(pi*x[1])*sin(pi*x[2]),
                        threshold=0,sign=1, mesh.type="unif",mesh.size=11,
                        intervals = matrix(c(1/2,5/2,1/2,5/2),nrow=2)))
```

`plot3d_mesh` *Plot a three dimensional mesh*

Description

Plot a three dimensional mesh

Usage

```
plot3d_mesh(
  mesh,
  engine3d = NULL,
  color.nodes = "black",
  color.mesh = "darkgray",
  alpha = 0.4,
  ...
)
```

Arguments

`mesh`	3-dimensional mesh to draw
`engine3d`	3d framework to use: 'rgl' if installed or 'scatterplot3d' (default)
`color.nodes`	color of the mesh nodes
`color.mesh`	color of the mesh elements
`alpha`	transparency of the mesh elements & nodes
`...`	optional arguments passed to plot function

Examples

```
if (identical(Sys.getenv("NOT_CRAN"), "true")) { # too long for CRAN on Windows

  plot3d_mesh(mesh_exsets(function(x) (0.5+x[1])^2+(-0.5+x[2])^2+(0.+x[3])^2,
                          threshold = .25,sign=-1, mesh.type="unif",
                          maxerror_f=1E-2,tol=1E-2, # faster display
                          intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2)),
                          engine3d='scatterplot3d')

  if (requireNamespace("rgl")) {
    plot3d_mesh(mesh_exsets(function(x) (0.5+x[1])^2+(-0.5+x[2])^2+(0.+x[3])^2,
                            threshold = .25,sign=-1, mesh.type="unif",
                            maxerror_f=1E-2,tol=1E-2, # faster display
                            intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2)),engine3d='rgl')
  }
}
```

`plot_mesh` *Plot a one dimensional mesh*

Description

Plot a one dimensional mesh

Usage

```
plot_mesh(
  mesh,
  y = 0,
  color.nodes = "black",
  color.mesh = "darkgray",
  alpha = 0.4,
  ...
)
```

Arguments

`mesh`	1-dimensional mesh to draw
`y`	ordinate value where to draw the mesh
`color.nodes`	color of the mesh nodes
`color.mesh`	color of the mesh elements
`alpha`	transparency of the mesh elements & nodes
`...`	optional arguments passed to plot function

Examples

```
plot_mesh(mesh_exsets(function(x) x, threshold=.51, sign=1, intervals=rbind(0,1)))
plot_mesh(mesh_exsets(function(x) (x-.5)^2, threshold=.1, sign=-1, intervals=rbind(0,1)))
```

`points_in.mesh` *Extract points of mesh which belong to the mesh triangulation (may not contain all points)*

Description

Extract points of mesh which belong to the mesh triangulation (may not contain all points)

Usage

```
points_in.mesh(mesh)
```

Arguments

`mesh`	mesh (list(p,tri,...) from geometry)

Value

points coordinates inside the mesh triangulation

`points_out.mesh` *Extract points of mesh which do not belong to the mesh triangulation (may not contain all points)*

Description

Extract points of mesh which do not belong to the mesh triangulation (may not contain all points)

Usage

```
points_out.mesh(mesh)
```

Arguments

`mesh`	(list(p,tri,...) from geometry)

Value

points coordinates outside the mesh triangulation root *One Dimensional Root (Zero) Finding*

Description

Search one root with given precision (on y). Iterate over uniroot as long as necessary.

Usage

```
root(
  f,
  lower,
  upper,
  maxerror_f = 1e-07,
  f_lower = f(lower, ...),
  f_upper = f(upper, ...),
  tol = .Machine$double.eps^0.25,
  convexity = FALSE,
  rec = 0,
  max.rec = NA,
  ...
)
```

Arguments

`f`	the function for which the root is sought.
`lower`	the lower end point of the interval to be searched.
`upper`	the upper end point of the interval to be searched.
`maxerror_f`	the maximum error on f evaluation (iterates over uniroot to converge).
`f_lower`	the same as f(lower).
`f_upper`	the same as f(upper).
`tol`	the desired accuracy (convergence tolerance on f arg).
`convexity`	the learned convexity factor of the function, used to reduce the boundaries for uniroot.
`rec`	counter of recursive level.
`max.rec`	maximal number of recursive level before failure (stop).
`...`	additional named or unnamed arguments to be passed to f.

Author(s)

Yann Richet, IRSN

Examples

```
f=function(x) {cat("f");1-exp(x)}; f(root(f,lower=-1,upper=2))
f=function(x) {cat("f");exp(x)-1}; f(root(f,lower=-1,upper=2))

.f = function(x) 1-exp(1*x)
f=function(x) {cat("f");y=.f(x);points(x,y,pch=20,col=rgb(0,0,0,.2));y}
plot(.f,xlim=c(-1,2)); f(root(f,lower=-1,upper=2))

.f = function(x) exp(10*x)-1
f=function(x) {cat("f");y=.f(x);points(x,y,pch=20);y}
plot(.f,xlim=c(-1,2)); f(root(f,lower=-1,upper=2))

.f = function(x) exp(100*x)-1
f=function(x) {cat("f");y=.f(x);points(x,y,pch=20);y}
plot(.f,xlim=c(-1,2)); f(root(f,lower=-1,upper=2))

f=function(x) {cat("f");exp(100*x)-1}; f(root(f,lower=-1,upper=2))

## Not run:

  # Quite hard functions to find roots

  ## Increasing function
  ## convex
  n.f=0
  .f = function(x) exp(10*x)-1
  f=function(x) {n.f<<-n.f+1;y=.f(x);points(x,y,pch=20);y}
  plot(.f,xlim=c(-.1,.2)); f(root(f,lower=-1,upper=2))
  print(n.f)
  ## non-convex
  n.f=0
  .f = function(x) 1-exp(-10*x)
  f=function(x) {n.f<<-n.f+1;y=.f(x);points(x,y,pch=20);y}
  plot(.f,xlim=c(-.1,.2)); f(root(f,lower=-1,upper=2))
  print(n.f)

  # ## Decreasing function
  # ## non-convex
  n.f=0
  .f = function(x) 1-exp(10*x)
  f=function(x) {n.f<<-n.f+1;y=.f(x);points(x,y,pch=20,col=rgb(0,0,0,.2));y}
  plot(.f,xlim=c(-.1,.2)); f(root(f,lower=-1,upper=2))
  print(n.f)
  # ## convex
  n.f=0
  .f = function(x) exp(-10*x)-1
  f=function(x) {n.f<<-n.f+1;y=.f(x);points(x,y,pch=20,col=rgb(0,0,0,.2));y}
  plot(.f,xlim=c(-.1,.2)); f(root(f,lower=-1,upper=2))
  print(n.f)

## End(Not run)
```

roots *One Dimensional Multiple Roots (Zero) Finding*

Description

Search multiple roots of 1D function, sampled/splitted by a (1D) mesh

Usage

```
roots(
  f,
  vectorized = FALSE,
  interval,
  maxerror_f = 1e-07,
  split = "seq",
  split.size = 11,
  tol = .Machine$double.eps^0.25,
  .lapply = parallel::mclapply,
  ...
)
```

Arguments

`f`	Function to find roots
`vectorized`	boolean: is f already vectorized ? (default: FALSE) or if function: vectorized version of f.
`interval`	bounds to inverse in
`maxerror_f`	the maximum error on f evaluation (iterates over uniroot to converge).
`split`	function or "unif" or "seq" (default) to preform interval partition
`split.size`	number of parts to perform uniroot inside
`tol`	the desired accuracy (convergence tolerance on f arg).
`.lapply`	control the loop/vectorization over different roots (defaults to multicore apply).
`...`	additional named or unnamed arguments to be passed to f.

Value

array of x, so f(x)=target

Examples

```
roots(sin,interval=c(pi/2,5*pi/2))
roots(sin,interval=c(pi/2,1.5*pi/2))

f=function(x)exp(x)-1;
f(roots(f,interval=c(-1,2)))
```

```
f=function(x)exp(1000*x)-1;
f(roots(f,interval=c(-1,2)))
```

`roots_mesh` *Multi Dimensional Multiple Roots (Zero) Finding, sampled by a mesh*

Description

Multi Dimensional Multiple Roots (Zero) Finding, sampled by a mesh

Usage

```
roots_mesh(
  f,
  vectorized = FALSE,
  intervals,
  mesh.type = "seq",
  mesh.sizes = 11,
  maxerror_f = 1e-07,
  tol = .Machine$double.eps^0.25,
  ...
)
```

Arguments

`f`	Function (one or more dimensions) to find roots of
`vectorized`	is f already vectorized ? (default: no)
`intervals`	bounds to inverse in, each column contains min and max of each dimension
`mesh.type`	function or "unif" or "seq" (default) to preform interval partition
`mesh.sizes`	number of parts for mesh (duplicate for each dimension if using "seq")
`maxerror_f`	the maximum error on f evaluation (iterates over uniroot to converge).
`tol`	the desired accuracy (convergence tolerance on f arg).
`...`	Other args for f

Value

matrix of x, so f(x)=0

Examples

```
roots_mesh(function(x) x-.51, intervals=rbind(0,1))
roots_mesh(function(x) sum(x)-.51, intervals=cbind(rbind(0,1),rbind(0,1)))
roots_mesh(sin,intervals=c(pi/2,5*pi/2))
roots_mesh(f = function(x) sin(pi*x[1])*sin(pi*x[2]),
          intervals = matrix(c(1/2,5/2,1/2,5/2),nrow=2))

r = roots_mesh(f = function(x) (0.25+x[1])^2+(0.5+x[2])^2 - .25,
intervals=matrix(c(-1,1,-1,1),nrow=2), mesh.size=5)
plot(r,xlim=c(-1,1),ylim=c(-1,1))

r = roots_mesh(function(x) (0.5+x[1])^2+(-0.5+x[2])^2+(0.+x[3])^2 - .5,
              mesh.sizes = 11,
              intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2))
scatterplot3d::scatterplot3d(r,xlim=c(-1,1),ylim=c(-1,1),zlim=c(-1,1))

roots_mesh(function(x)exp(x)-1,intervals=c(-1,2))
roots_mesh(function(x)exp(1000*x)-1,intervals=c(-1,2))
```

sectionview.function *Plot a section view of a prediction model or function, including design points if available.*

Description

Plot a section view of a prediction model or function, including design points if available.

Usage

```
## S3 method for class '`function`'
sectionview(
  fun,
  vectorized = FALSE,
  dim = NULL,
  center = NULL,
  lty_center = 2,
  col_center = "black",
  axis = NULL,
  npoints = 101,
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  col_fading_interval = 0.5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'matrix'
sectionview(
  X,
  y,
  center = NULL,
  lty_center = 2,
  col_center = "black",
  axis = NULL,
  col_points = if (!is.null(col)) col else "red",
  col = NULL,
  col_fading_interval = 0.5,
  bg_fading = 5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'character'
sectionview(eval_str, axis = NULL, mfrow = NULL, ...)

## S3 method for class 'km'
sectionview(
  km_model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 101,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'Kriging'
sectionview(
  Kriging_model,
  center = NULL,
  axis = NULL,
  npoints = 101,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'NuggetKriging'
sectionview(
  NuggetKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 101,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)
```

```
## S3 method for class 'NoiseKriging'
sectionview(
  NoiseKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 101,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'glm'
sectionview(
  glm_model,
  center = NULL,
  axis = NULL,
  npoints = 101,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'list'
sectionview(
  modelFit_model,
  center = NULL,
  axis = NULL,
  npoints = 101,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  bg_fading = 5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

sectionview(...)
```

Arguments

`fun`	a function or 'predict()'-like function that returns a simple numeric, or an interval, or mean and standard error: list(mean=...,se=...).
`vectorized`	is fun vectorized?
`dim`	input variables dimension of the model or function.
`center`	optional coordinates (as a list or data frame) of the center of the section view if the model's dimension is > 2.
`lty_center`	line type for the section center of the plot (if any).
`col_center`	color for the section center of the plot (if any).
`axis`	optional matrix of 2-axis combinations to plot, one by row. The value `NULL` leads to all possible combinations i.e. `choose(D, 2)`.
`npoints`	an optional number of points to discretize plot of response surface and uncertainties.
`col_fun`	color of the function plot.
`col`	color of the object (use col_* for specific objects).
`col_fading_interval`	an optional factor of alpha (color channel) fading used to plot confidence intervals.
`mfrow`	an optional list to force `par(mfrow = ...)` call. The default value `NULL` is automatically set for compact view.
`Xlab`	an optional list of string to overload names for X.
`ylab`	an optional string to overload name for y.
`Xlim`	an optional list to force x range for all plots. The default value `NULL` is automatically set to include all design points.
`ylim`	an optional list to force y range for all plots.

`title`	an optional overload of main title.
`add`	to print graphics on an existing window.
`...`	arguments of the `sectionview.km`, `sectionview.glm`, `sectionview.Kriging` or `sectionview.function` function
`X`	the matrix of input design.
`y`	the array of output values (two columns means an interval).
`col_points`	color of points.
`bg_fading`	an optional factor of alpha (color channel) fading used to plot design points outside from this section.
`eval_str`	the expression to evaluate in each subplot.
`km_model`	an object of class `"km"`.
`type`	the kriging type to use for model prediction.
`conf_level`	an optional list of confidence intervals to display.
`conf_fading`	an optional factor of alpha (color channel) fading used to plot confidence intervals.
`Kriging_model`	an object of class `"Kriging"`.
`NuggetKriging_model`	an object of class `"Kriging"`.
`NoiseKriging_model`	an object of class `"Kriging"`.
`glm_model`	an object of class `"glm"`.
`modelFit_model`	an object returned by DiceEval::modelFit.

Details

If available, experimental points are plotted with fading colors. Points that fall in the specified section (if any) have the color specified `col_points` while points far away from the center have shaded versions of the same color. The amount of fading is determined using the Euclidean distance between the plotted point and `center`.

Author(s)

Yann Richet, IRSN

See Also

`sectionview.function` for a section plot, and `sectionview3d.function` for a 2D section plot.

`sectionview.matrix` for a section plot, and `sectionview3d.matrix` for a 2D section plot.

`sectionview.matrix` for a section plot, and `sectionview3d.matrix` for a 2D section plot.

`sectionview.km` for a section plot, and `sectionview3d.km` for a 2D section plot.

`sectionview.Kriging` for a section plot, and `sectionview3d.Kriging` for a 2D section plot.

`sectionview.NuggetKriging` for a section plot, and `sectionview3d.NuggetKriging` for a 2D section plot.

sectionview.NoiseKriging for a section plot, and sectionview3d.NoiseKriging for a 2D section plot.

sectionview.glm for a section plot, and sectionview3d.glm for a 2D section plot.

sectionview.glm for a section plot, and sectionview3d.glm for a 2D section plot.

Examples

```
x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2 + rnorm(15)
model <- lm(y ~ x1 + x2)

sectionview(function(x) sum(x),
                     dim=2, center=c(0,0), Xlim=cbind(range(x1),range(x2)), col='black')

sectionview(function(x) {
                      x = as.data.frame(x)
                      colnames(x) <- all.vars(model$call)[-1]
                      p = predict.lm(model, newdata=x, se.fit=TRUE)
                      cbind(p$fit-1.96 * p$se.fit, p$fit+1.96 * p$se.fit)
                    }, vectorized=TRUE, dim=2, center=c(0,0),
              Xlim=cbind(range(x1),range(x2)), add=TRUE)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

sectionview(X,y, center=c(.5,.5))

x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2^2 + rnorm(15)
model <- glm(y ~ x1 + I(x2^2))

sectionview(model, center=c(.5,.5))

sectionview("abline(h=5)")
if (requireNamespace("DiceKriging")) { library(DiceKriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- km(design = X, response = y, covtype="matern3_2")

sectionview(model, center=c(.5,.5))

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- Kriging(X = X, y = y, kernel="matern3_2")

sectionview(model, center=c(.5,.5))

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)

model <- NuggetKriging(X = X, y = y, kernel="matern3_2")

sectionview(model, center=c(.5,.5))

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)

model <- NoiseKriging(X = X, y = y, kernel="matern3_2", noise=rep(5^2,15))

sectionview(model, center=c(.5,.5))

}

x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2^2 + rnorm(15)
model <- glm(y ~ x1 + I(x2^2))

sectionview(model, center=c(.5,.5))

if (requireNamespace("DiceEval")) { library(DiceEval)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- modelFit(X, y, type = "StepLinear")

sectionview(model, center=c(.5,.5))

}

## A 2D example - Branin-Hoo function
sectionview(branin, center= c(.5,.5), col='black')

## Not run:
```

```
## a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
design.fact <- expand.grid(seq(0, 1, length = 4), seq(0, 1, length = 4))
design.fact <- data.frame(design.fact); names(design.fact) <- c("x1", "x2")
y <- branin(design.fact); names(y) <- "y"

if (requireNamespace("DiceKriging")) { library(DiceKriging)
## model: km
model <- DiceKriging::km(design = design.fact, response = y)
sectionview(model, center= c(.5,.5))
sectionview(branin, center= c(.5,.5), col='red', add=TRUE)
}

if (requireNamespace("rlibkriging")) { library(rlibkriging)
## model: Kriging
model <- Kriging(X = as.matrix(design.fact), y = as.matrix(y), kernel="matern3_2")
sectionview(model, center= c(.5,.5))
sectionview(branin, center= c(.5,.5), col='red', add=TRUE)
}

## model: glm
model <- glm(y ~ 1+ x1 + x2 + I(x1^2) + I(x2^2) + x1*x2, data=cbind(y,design.fact))
sectionview(model, center= c(.5,.5))
sectionview(branin, center= c(.5,.5), col='red', add=TRUE)

if (requireNamespace("DiceEval")) { library(DiceEval)
## model: StepLinear
model <- modelFit(design.fact, y, type = "StepLinear")
sectionview(model, center= c(.5,.5))
sectionview(branin, center= c(.5,.5), col='red', add=TRUE)
}

## End(Not run)
```

sectionview3d.function

Plot a contour view of a prediction model or function, including design points if available.

Description

Plot a contour view of a prediction model or function, including design points if available.

Usage

```
## S3 method for class '`function`'
sectionview3d(
  fun,
  vectorized = FALSE,
  dim = NULL,
  center = NULL,
  axis = NULL,
  npoints = 21,
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  col_fading_interval = 0.5,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'matrix'
sectionview3d(
  X,
  y,
  center = NULL,
  axis = NULL,
  col_points = if (!is.null(col)) col else "red",
  col = NULL,
  col_fading_interval = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'character'
sectionview3d(eval_str, axis = NULL, mfrow = NULL, ...)

## S3 method for class 'km'
sectionview3d(
  km_model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 21,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'Kriging'
sectionview3d(
  Kriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'NuggetKriging'
sectionview3d(
  NuggetKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'NoiseKriging'
sectionview3d(
  NoiseKriging_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'glm'
sectionview3d(
  glm_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  conf_level = 0.95,
  conf_fading = 0.5,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'list'
sectionview3d(
  modelFit_model,
  center = NULL,
  axis = NULL,
  npoints = 21,
  col_points = if (!is.null(col)) col else "red",
  col_fun = if (!is.null(col)) col else "blue",
  col = NULL,
  bg_fading = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

sectionview3d(...)
```

Arguments

`fun`	a function or 'predict()'-like function that returns a simple numeric, or an interval, or mean and standard error: list(mean=...,se=...).
`vectorized`	is fun vectorized?
`dim`	input variables dimension of the model or function.
`center`	optional coordinates (as a list or data frame) of the center of the section view if the model's dimension is > 2.
`axis`	optional matrix of 2-axis combinations to plot, one by row. The value `NULL` leads to all possible combinations i.e. `choose(D, 2)`.

`npoints`	an optional number of points to discretize plot of response surface and uncertainties.
`col_fun`	color of the function plot.
`col`	color of the object (use col_* for specific objects).
`col_fading_interval`	an optional factor of alpha (color channel) fading used to plot confidence intervals.
`mfrow`	an optional list to force `par(mfrow = ...)` call. The default value `NULL` is automatically set for compact view.
`Xlab`	an optional list of string to overload names for X.
`ylab`	an optional string to overload name for y.
`Xlim`	an optional list to force x range for all plots. The default value `NULL` is automatically set to include all design points (and their 1-99 percentiles).
`ylim`	an optional list to force y range for all plots. The default value `NULL` is automatically set to include all design points (and their 1-99 percentiles).
`title`	an optional overload of main title.
`add`	to print graphics on an existing window.
`engine3d`	3D view package to use. "rgl" if available, otherwise "scatterplot3d" by default.
`...`	arguments of the `sectionview3d.km`, `sectionview3d.glm`, `sectionview3d.Kriging` or `sectionview3d.function` function
`X`	the matrix of input design.
`y`	the array of output values (two columns means an interval).
`col_points`	color of points.
`bg_fading`	an optional factor of alpha (color channel) fading used to plot design points outside from this section.
`eval_str`	the expression to evaluate in each subplot.
`km_model`	an object of class `"km"`.
`type`	the kriging type to use for model prediction.
`conf_level`	an optional list of confidence intervals to display.
`conf_fading`	an optional factor of alpha (color channel) fading used to plot confidence intervals.
`Kriging_model`	an object of class `"Kriging"`.
`NuggetKriging_model`	an object of class `"Kriging"`.
`NoiseKriging_model`	an object of class `"Kriging"`.
`glm_model`	an object of class `"glm"`.
`modelFit_model`	an object returned by DiceEval::modelFit.

Details

If available, experimental points are plotted with fading colors. Points that fall in the specified section (if any) have the color specified `col_points` while points far away from the center have shaded versions of the same color. The amount of fading is determined using the Euclidean distance between the plotted point and `center`.

Author(s)

Yann Richet, IRSN

See Also

`sectionview.function` for a section plot, and `sectionview3d.function` for a 2D section plot.

`sectionview.matrix` for a section plot, and `sectionview3d.matrix` for a 2D section plot.

`sectionview3d.matrix` for a section plot.

`sectionview.km` for a section plot, and `sectionview3d.km` for a 2D section plot.

`sectionview.Kriging` for a section plot, and `sectionview3d.Kriging` for a 2D section plot.

`sectionview.NuggetKriging` for a section plot, and `sectionview3d.NuggetKriging` for a 2D section plot.

`sectionview.NoiseKriging` for a section plot, and `sectionview3d.NoiseKriging` for a 2D section plot.

`sectionview.glm` for a section plot, and `sectionview3d.glm` for a 2D section plot.

`sectionview.glm` for a section plot, and `sectionview3d.glm` for a 2D section plot.

Examples

```
x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2 + rnorm(15)
model <- lm(y ~ x1 + x2)

DiceView:::open3d();
DiceView:::plot3d(x1,x2,y)
sectionview3d(function(x) sum(x),
                     dim=2, Xlim=cbind(range(x1),range(x2)), add=TRUE, col='black')

sectionview3d(function(x) {
                      x = as.data.frame(x)
                      colnames(x) <- all.vars(model$call)[-1]
                      p = predict.lm(model, newdata=x, se.fit=TRUE)
                      list(mean=p$fit, se=p$se.fit)
                    }, vectorized=TRUE, dim=2,
              Xlim=cbind(range(x1),range(x2)), add=TRUE)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)
```

```
sectionview3d(X, y)

x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2^2 + rnorm(15)
model <- glm(y ~ x1 + I(x2^2))

sectionview3d(model)

sectionview3d("abline(h=0.25,col='red')")
if (requireNamespace("DiceKriging")) { library(DiceKriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- km(design = X, response = y, covtype="matern3_2")

sectionview3d(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- Kriging(X = X, y = y, kernel="matern3_2")

sectionview3d(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)

model <- NuggetKriging(X = X, y = y, kernel="matern3_2")

sectionview3d(model)

}

if (requireNamespace("rlibkriging")) { library(rlibkriging)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin) + 5*rnorm(15)

model <- NoiseKriging(X = X, y = y, kernel="matern3_2", noise=rep(5^2,15))

sectionview3d(model)
```

```
}

x1 <- rnorm(15)
x2 <- rnorm(15)

y <- x1 + x2^2 + rnorm(15)
model <- glm(y ~ x1 + I(x2^2))

sectionview3d(model)

if (requireNamespace("DiceEval")) { library(DiceEval)

X = matrix(runif(15*2),ncol=2)
y = apply(X,1,branin)

model <- modelFit(X, y, type = "StepLinear")

sectionview3d(model)

}

## A 2D example - Branin-Hoo function
sectionview3d(branin, dim=2, col='black')

## Not run:
## a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
design.fact <- expand.grid(seq(0, 1, length = 4), seq(0, 1, length = 4))
design.fact <- data.frame(design.fact); names(design.fact) <- c("x1", "x2")
y <- branin(design.fact); names(y) <- "y"

if (requireNamespace("DiceKriging")) { library(DiceKriging)
## model: km
model <- DiceKriging::km(design = design.fact, response = y)
sectionview3d(model)
sectionview3d(branin, dim=2, col='red', add=TRUE)
}

if (requireNamespace("rlibkriging")) { library(rlibkriging)
## model: Kriging
model <- Kriging(X = as.matrix(design.fact), y = as.matrix(y), kernel="matern3_2")
sectionview3d(model)
sectionview3d(branin, dim=2, col='red', add=TRUE)
}

## model: glm
model <- glm(y ~ 1+ x1 + x2 + I(x1^2) + I(x2^2) + x1*x2, data=cbind(y,design.fact))
sectionview3d(model)
sectionview3d(branin, dim=2, col='red', add=TRUE)

if (requireNamespace("DiceEval")) { library(DiceEval)
## model: StepLinear
model <- modelFit(design.fact, y, type = "StepLinear")
```

```
sectionview3d(model)
sectionview3d(branin, dim=2, col='red', add=TRUE)
}

## End(Not run)
```

`Vectorize.function` *Vectorize a multidimensional Function*

Description

Vectorize a d-dimensional (input) function, in the same way that base::Vectorize for 1-dimensional functions.

Usage

```
Vectorize.function(
  fun,
  dim,
  .combine = rbind,
  .lapply = parallel::mclapply,
  ...
)
```

Arguments

fun	'dim'-dimensional function to Vectorize
dim	dimension of input arguments of fun
.combine	how to combine results (default using c(.))
.lapply	how to vectorize FUN call (default is parallel::mclapply)
...	optional args to pass to 'Apply.function()', including .combine, .lapply, or optional args passed to 'fun'.

Value

a vectorized function (to be called on matrix argument, on each row)

Examples

```
f = function(x)x[1]+1; f(1:10); F = Vectorize.function(f,1);
F(1:10); #F = Vectorize(f); F(1:10);

f2 = function(x)x[1]+x[2]; f2(1:10); F2 = Vectorize.function(f2,2);
F2(cbind(1:10,11:20));

f3 = function(x)list(mean=x[1]+x[2],se=x[1]*x[2]); f3(1:10); F3 = Vectorize.function(f3,2);
F3(cbind(1:10,11:20));
```

Index