Package ‘DiceView’

July 3, 2020

Title Methods for Visualization of Computer Experiments Design and Surrogate

Version 2.0-0

Date 2020-07-01

Author Yann Richet, Yves Deville, Clement Chevalier

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, rgl, scatterplot3d, DiceKriging, DiceEval

Imports DiceDesign, R.cache, geometry

License GPL-3

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

Repository CRAN

RoxygenNote 7.1.1

NeedsCompilation no

Date/Publication 2020-07-03 08:40:02 UTC

R topics documented:

`Apply.fun` *Apply Functions Over Array Margins, using custom vecorization (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.fun(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.fun(sum,X)

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

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

`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 *Plot a contour view of a kriging or modelPredict model including design points, or a function.*

Description

Plot a contour view of a kriging or modelPredict model. It is useful for a better understanding of a model behaviour.

Usage

```
contourview(model, ...)
```

Arguments

model	an object of class "km", a list that can be used in a "modelPredict" call, or a function.
...	other arguments of the `contourview.km`, `contourview.list` or `contourview.function` function

Examples

```
## A 2D example - Branin-Hoo function
## 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)

## kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
m1 <- km(design = design.fact, response = y)

contourview(m1)

contourview(branin, dim = 2, add=TRUE)
```

contourview.function *Plot a contour view of a function.*

Description

Plot one section view per dimension of a function thus providing a better understanding of the model behaviour.

Plot a 3-D view of a function. Provide a better understanding of the model behaviour.

Usage

```
## S3 method for class '`function`'
contourview(
  model,
  dim = ifelse(is.null(center), 2, length(center)),
  center = NULL,
  axis = NULL,
  npoints = 20,
  nlevels = 10,
  col = "blue",
  filled = FALSE,
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = c(0, 1),
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class '`function`'
sectionview(
  model,
  dim = ifelse(is.null(center), 1, length(center)),
  center = NULL,
  axis = NULL,
  npoints = 100,
  col_surf = "blue",
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
```

```
  xlim = c(0, 1),
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class '`function`'
sectionview3d(
  model,
  dim = ifelse(is.null(center), 2, length(center)),
  center = NULL,
  axis = NULL,
  npoints = 20,
  col = "blue",
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = c(0, 1),
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S4 method for signature '`function`'
sectionview(
  model,
  dim,
  center = NULL,
  axis = NULL,
  npoints = 100,
  col = "blue",
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = c(0, 1),
  ylim = NULL,
  title = NULL,
  ...
)

## S4 method for signature '`function`'
sectionview3d(
  model,
```

```
  dim,
  center = NULL,
  axis = NULL,
  npoints = 20,
  col = "blue",
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = c(0, 1),
  ylim = NULL,
  title = NULL,
  ...
)

## S4 method for signature '`function`'
contourview(
  model,
  dim,
  center = NULL,
  axis = NULL,
  npoints = 20,
  col = "blue",
  nlevels = 10,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = c(0, 1),
  ylim = NULL,
  title = NULL,
  ...
)
```

Arguments

`model`	an object of class `"function"`.
`dim`	the dimension of fun arguments.
`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.
`nlevels`	number of contour levels to display.
`col`	color for the surface.
`filled`	use filled.contour

mfrow	an optional list to force par(mfrow = ...) call. The default value NULL is automatically set for compact view.
Xname	an optional list of string to overload names for X.
yname	an optional string to overload name for y.
Xscale	an optional factor to scale X.
yscale	an optional factor to scale y.
xlim	a list to give x range for all plots.
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.
...	further arguments passed to the first call of plot3d.
col_surf	color for the section.
function	function, taken as model

Details

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. The variables chosen with their number are to be found in the X slot of the model. Thus they are 'spatial dimensions' but not 'trend variables'.

A multiple rows/columns plot is produced.

Author(s)

Yann Richet, IRSN

Yann Richet, IRSN

Yann Richet, IRSN

See Also

See sectionview3d.

The function sectionview3d produces a 3D version.

sectionview

Examples

```
## A 2D example - Branin-Hoo function.
contourview(branin,dim = 2)
## A 2D example - Branin-Hoo function.
sectionview(branin,center=c(.5,.5))
## A 2D example - Branin-Hoo function.
sectionview3d(branin,dim = 2)
```

contourview.km *Plot a contour view of a kriging model, including design points*

Description

Plot a contour view of a kriging model: mean response surface, fitted points and confidence surfaces. Provide a better understanding of the kriging model behaviour.

Plot one section view per dimension of a kriging model thus providing a better understanding of the model behaviour including uncertainty.

Plot a 3-D view of a kriging model: mean response surface, fitted points and confidence surfaces. Provide a better understanding of the kriging model behaviour.

Usage

```
## S3 method for class 'km'
contourview(
  model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 20,
  nlevels = 10,
  col_points = "red",
  col_surf = "blue",
  filled = FALSE,
  bg_blend = 1,
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'km'
sectionview(
  model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 100,
  col_points = "red",
```

```
  col_surf = "blue",
  conf_lev = c(0.5, 0.8, 0.9, 0.95, 0.99),
  conf_blend = NULL,
  bg_blend = 5,
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'km'
sectionview3d(
  model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 20,
  col_points = "red",
  col_surf = "blue",
  col_needles = NA,
  conf_lev = c(0.95),
  conf_blend = NULL,
  bg_blend = 5,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S4 method for signature 'km'
sectionview(
  model,
  type = "UK",
  center = NULL,
  npoints = 100,
  col_points = "red",
  col_surf = "blue",
```

```
  conf_lev = c(0.5, 0.8, 0.9, 0.95, 0.99),
  conf_blend = NULL,
  bg_blend = 5,
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  ...
)

## S4 method for signature 'km'
sectionview3d(
  model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 20,
  col_points = "red",
  col_surf = "blue",
  col_needles = NA,
  conf_lev = c(0.95),
  conf_blend = NULL,
  bg_blend = 5,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  ...
)

## S4 method for signature 'km'
contourview(
  model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 20,
  col_points = "red",
  col_surf = "blue",
  bg_blend = 1,
  nlevels = 10,
```

```
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  ...
)
```

Arguments

`model`	an object of class "km".
`type`	the kriging type to use for model prediction.
`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.
`nlevels`	number of contour levels to display.
`col_points`	color of points.
`col_surf`	color for the surface.
`filled`	use filled.contour
`bg_blend`	an optional factor of alpha (color channel) blending used to plot design points outside from this section.
`mfrow`	an optional list to force `par(mfrow = ...)` call. The default value `NULL` is automatically set for compact view.
`Xname`	an optional list of string to overload names for X.
`yname`	an optional string to overload name for y.
`Xscale`	an optional factor to scale X.
`yscale`	an optional factor to scale 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. 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.
`...`	further arguments passed to the first call of `plot3d`.
`conf_lev`	an optional list of confidence interval values to display.
`conf_blend`	an optional factor of alpha (color channel) blending used to plot confidence intervals.

`col_needles`	color of "needles" for the points. The default `NA` corresponds to no needle plotted. When a valid color is given, needles are plotted using the same fading mechanism as for points.
`km`	kriging model

Details

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`. The variables chosen with their number are to be found in the `X` slot of the model. Thus they are 'spatial dimensions' but not 'trend variables'.

A multiple rows/columns plot is produced. 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`.

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`. The variables chosen with their number are to be found in the `X` slot of the model. Thus they are 'spatial dimensions' but not 'trend variables'.

Note

The confidence bands are computed using normal quantiles and the standard error given by `predict.km`.

The confidence bands are computed using normal quantiles and the standard error given by `predict.km`.

Author(s)

Yann Richet, IRSN

Yann Richet, IRSN

Yann Richet, IRSN

See Also

See `sectionview3d.km` and the `km` function in the **DiceKriging** package.

The function `sectionview3d.km` produces a 3D version. For more information on the `km` class, see the `km` function in the **DiceKriging** package.

See `sectionview.km` and the `km` function in the **DiceKriging** package.

Examples

```
## A 2D example - Branin-Hoo function. See DiceKriging package manual
## 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)

## kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
if (!exists("m1"))
m1 <- km(design = design.fact, response = y)

## the same as contourview.km
contourview(m1)

## change colors
contourview(m1, col_points = "firebrick", col_surf = "SpringGreen2")

## change colors,  use finer grid and add needles
contourview(m1, npoints = c(50, 30), col_points = "orange",
col_surf = "SpringGreen2")

## Display reference function
contourview(branin,dim=2,add=TRUE,col='red')
## A 2D example - Branin-Hoo function
## 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)

## kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
if (!exists("m1"))
m1 <- km(design = design.fact, response = y)

sectionview(m1, center = c(.333, .333))

## Display reference function
sectionview(branin,dim=2,center=c(.333, .333),add=TRUE,col='red')
## A 2D example - Branin-Hoo function. See DiceKriging package manual
## 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)

## kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
if (!exists("m1"))
m1 <- km(design = design.fact, response = y)

## the same as sectionview3d.km
sectionview3d(m1)

## Not run:
## change colors
sectionview3d(m1, col_points = "firebrick", col_surf = "SpringGreen2")

## change colors,  use finer grid and add needles
sectionview3d(m1, npoints = c(50, 30), col_points = "orange",
```

```
  col_surf = "SpringGreen2", col_needles = "firebrick")

## End(Not run)
```

contourview.list *Plot a contour view of a model, including design points*

Description

Plot a contour view of a model, thus providing a better understanding of its behaviour.

Plot one section view per dimension of a surrogate model. It is useful for a better understanding of a model behaviour.

Plot a 3-D view of a model, thus providing a better understanding of its behaviour.

Usage

```
## S3 method for class 'list'
contourview(
  model,
  center = NULL,
  axis = NULL,
  npoints = 20,
  nlevels = 10,
  col_points = "red",
  col_surf = "blue",
  filled = FALSE,
  bg_blend = 1,
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'list'
sectionview(
  model,
  center = NULL,
  axis = NULL,
  npoints = 100,
  col_points = "red",
  col_surf = "blue",
```

```
  bg_blend = 5,
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'list'
sectionview3d(
  model,
  center = NULL,
  axis = NULL,
  npoints = 20,
  col_points = "red",
  col_surf = "blue",
  col_needles = NA,
  bg_blend = 5,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S4 method for signature 'list'
sectionview(
  model,
  center = NULL,
  npoints = 100,
  col_points = "red",
  col_surf = "blue",
  bg_blend = 5,
  mfrow = NULL,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
```

```
  ylim = NULL,
  title = NULL,
  ...
)

## S4 method for signature 'list'
sectionview3d(
  model,
  center = NULL,
  axis = NULL,
  npoints = 20,
  col_points = "red",
  col_surf = "blue",
  bg_blend = 5,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  ...
)

## S4 method for signature 'list'
contourview(
  model,
  center = NULL,
  axis = NULL,
  npoints = 20,
  col_points = "red",
  col_surf = "blue",
  bg_blend = 1,
  nlevels = 10,
  Xname = NULL,
  yname = NULL,
  Xscale = 1,
  yscale = 1,
  xlim = NULL,
  ylim = NULL,
  title = NULL,
  ...
)
```

Arguments

`model`	a list that can be used in the `modelPredict` function of the **DiceEval** package.
`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.
`nlevels`	number of contour levels to display.
`col_points`	color of points.
`col_surf`	color for the surface.
`filled`	use filled.contour
`bg_blend`	an optional factor of alpha (color channel) blending used to plot design points outside from this section.
`mfrow`	an optional list to force `par(mfrow = ...)` call. Default (NULL value) is automatically set for compact view.
`Xname`	an optional list of string to overload names for X.
`yname`	an optional string to overload name for y.
`Xscale`	an optional factor to scale X.
`yscale`	an optional factor to scale 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. 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.
`...`	optional arguments passed to the first call of `plot3d`.
`col_needles`	color of "needles" for the points. The default `NA` corresponds to no needle plotted. When a valid color is given, needles are plotted using the same fading mechanism as for points.
`list`	DiceEval model

Details

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`. The variables chosen with their number are to be found in the `data$X` element of the model. Thus they are original data variables but not trend variables that may have been created using the model's formula.

A multiple rows/columns plot is produced. 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`.

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`. The variables chosen with their number are to be found in the `data$X` element of the model. Thus they are original data variables but not trend variables that may have been created using the model's formula

Author(s)

Yann Richet, IRSN

Yann Richet, IRSN

Yann Richet, IRSN

See Also

`sectionview.list` for a 2D plot, and the `modelPredict` function in the **DiceEval** package. The `sectionview3d.km` produces a similar plot for `km` objects.

See `sectionview3d.list` for a 3d version, and the `modelPredict` function in the **DiceEval** package.

`sectionview.list` for a 2D plot, and the `modelPredict` function in the **DiceEval** package. The `sectionview3d.km` produces a similar plot for `km` objects.

Examples

```
## A 2D example - Branin-Hoo function
## 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)

## linear model
m1 <- modelFit(design.fact, y[[1]], type = "Linear", formula = "Y~.")

## the same as sectionview3d.list
contourview(m1)
## A 2D example: Branin-Hoo function. See the DiceKriging package manual
## 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)

## linear model
m1 <- modelFit(design.fact, y[[1]], type = "Linear", formula = "Y~.")

sectionview(m1, center = c(.333,.333))
## A 2D example - Branin-Hoo function
## 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)

## linear model
m1 <- modelFit(design.fact, y[[1]], type = "Linear", formula = "Y~.")

## the same as sectionview3d.list
sectionview3d(m1)
```

---

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.fun` *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.fun(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.fun(f);
F(5); F(6); F(5)
```

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,
  f.vectorized = FALSE,
  threshold,
  sign,
  intervals,
  mesh = "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'
`f.vectorized`	is f already vectorized ? (default: no)
`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`	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`	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 mesh_roots(...) call

Examples

```
mesh_exsets(function(x) x, threshold=.51, sign=1, intervals=rbind(0,1))
mesh_exsets(function(x) x, threshold=.50000001, sign=1, intervals=rbind(0,1))
mesh_exsets(function(x) sum(x), threshold=.51,sign=1, intervals=cbind(rbind(0,1),rbind(0,1)))
mesh_exsets(sin,threshold=0,sign="sup",interval=c(pi/2,5*pi/2))
```

```
mesh_exsets(f = function(x) sin(pi*x[1])*sin(pi*x[2]),
            threshold=0,sign=1, intervals = matrix(c(1/2,5/2,1/2,5/2),nrow=2))

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))
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)))

## Not run:
e = mesh_exsets(function(x) (0.5+x[1])^2+(-0.5+x[2])^2+(0.+x[3])^2,
              threshold = .25,sign=-1, mesh="unif", mesh.sizes = 10,
              intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2))
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,]))

## End(Not run)
```

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

Description

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

Usage

```
mesh_roots(
  f,
  f.vectorized = FALSE,
  intervals,
  mesh = "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
`f.vectorized`	is f already vectorized ? (default: no)
`intervals`	bounds to inverse in, each column contains min and max of each dimension
`mesh`	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

```
mesh_roots(function(x) x-.51, intervals=rbind(0,1))
mesh_roots(function(x) sum(x)-.51, intervals=cbind(rbind(0,1),rbind(0,1)))
mesh_roots(sin,intervals=c(pi/2,5*pi/2))
mesh_roots(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 = mesh_roots(function(x) (0.25+x[1])^2+(0.5+x[2])^2-.25,
              intervals=matrix(c(-1,1,-1,1),nrow=2))
plot(r,xlim=c(-1,1),ylim=c(-1,1))

r = mesh_roots(function(x) (0.5+x[1])^2+(-0.5+x[2])^2+(0.+x[3])^2- .25,
              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))

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

---

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))
```

plot2d_mesh	*Plot a two dimensional mesh*

Description

Plot a two dimensional mesh

Usage

```
plot2d_mesh(mesh, color = "black", ...)
```

Arguments

mesh	2-dimensional mesh to draw
color	color of the mesh
...	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="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, view = "scatterplot3d", color = "black", ...)
```

Arguments

mesh	3-dimensional mesh to draw
view	3d framework to use: 'rgl' or 'scatterplot3d' (default)
color	color of the mesh
...	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="unif",mesh.size=11,
                        intervals = matrix(c(1/2,5/2,1/2,5/2),nrow=2)))

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="unif", mesh.sizes = 10,
                        intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2)))

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="unif", mesh.sizes = 10,
                        intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2)),mode='rgl')
```

`plot_mesh` *Plot a one dimensional mesh*

Description

Plot a one dimensional mesh

Usage

```
plot_mesh(mesh, y = 0, color = "black", ...)
```

Arguments

`mesh`	1-dimensional mesh to draw
`y`	ordinate value where to draw the mesh
`color`	color of the mesh
`...`	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 = 0,
  ...
)
```

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.
`...`	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))
```

`roots` *One Dimensional Multiple Roots (Zero) Finding*

Description

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

Usage

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

Arguments

`f`	Function to find roots
`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).
`...`	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)))
```

---

sectionview — *Plot a section view of a kriging or modelPredict model including design points, or a function.*

Description

Plot one section view per dimension of a kriging, `modelPredict` model or function. It is useful for a better understanding of a model behaviour (including uncertainty).

Usage

```
sectionview(model, ...)
```

Arguments

`model`	an object of class "`km`", a list that can be used in a "`modelPredict`" call, or a function.
`...`	other arguments of the `contourview.km`, `contourview.list` or `contourview.function` function

Examples

```
## A 2D example - Branin-Hoo function
## 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)

## kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
m1 <- km(design = design.fact, response = y)

sectionview(m1, center = c(.333, .333))

sectionview(branin, dim = 2, center = c(.333, .333), add=TRUE)
```

`sectionview3d` *Plot a 3-D (using RGL) view of a kriging or modelPredict model, including design points*

Description

Plot a 3-D view of a kriging or modelPredict model. It is useful for a better understanding of a model behaviour.

Usage

```
sectionview3d(model, ...)
```

Arguments

`model`	an object of class `"km"`, a list that can be used in a `"modelPredict"` call, or a function.
`...`	other arguments of the `contourview.km`, `contourview.list` or `contourview.function` function

Examples

```
## A 2D example - Branin-Hoo function
## 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)

## kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
m1 <- km(design = design.fact, response = y)

sectionview3d(m1)

sectionview3d(branin, dim = 2, add=TRUE)
```

`Vectorize.funD` *Vectorize a multidimensional Function*

Description

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

Usage

```
Vectorize.funD(fund, d, .apply = base::apply)
```

Arguments

`fund`	d-dimensional function to Vectorize
`d`	dimension of input arguments of fund
`.apply`	which vectorization to use (default is base::apply)

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.funD(f,1);
F(1:10); #F = Vectorize(f); F(1:10);

f2 = function(x)x[1]+x[2]; f2(1:10); F = Vectorize.funD(f2,2);
F(cbind(1:10,11:20)); #F = Vectorize(f); F(1:10);
```

Index