Package ‘DiceView’

January 15, 2024

Title Methods for Visualization of Computer Experiments Design and Surrogate
Version 2.2-0
Date 2024-01-15
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 7.2.3
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 2024-01-15 10:00:02 UTC

R topics documented:

Apply.function .............................................................. 2
are_in.mesh ............................................................... 3
branin ................................................................. 4
combn.design ............................................................ 4
contourview.function ................................................ 5
is_in.mesh ............................................................. 12
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

\[
X = \text{matrix}(\text{runif}(10), \text{ncol}=2);
\]
\[
\text{rowSums}(X) == \text{apply}(X, 1, \text{sum})
\]
\[
\text{apply}(X, 1, \text{sum}) == \text{Apply.function}(\text{sum}, X)
\]

\[
X = \text{matrix}(\text{runif}(10), \text{ncol}=1)
\]
\[
\text{rowSums}(X) == \text{apply}(X, 1, \text{sum})
\]
\[
\text{apply}(X, 1, \text{sum}) == \text{Apply.function}(\text{sum}, X)
\]

\[
X = \text{matrix}(\text{runif}(10), \text{ncol}=2)
\]
\[
f = \text{function}(X) X[1]/X[2]
\]
\[
\text{apply}(X, 1, f) == \text{Apply.function}(f, X)
\]

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

\[
X = \text{matrix}(\text{runif}(100), \text{ncol}=2);
\]
\[
\text{inside} = \text{are_in.mesh}(X, \text{mesh=geometry::delaunayn(matrix(c(0,0,1,1,0,0),ncol=2),output.options =TRUE))}
\]
\[
\text{print}(\text{inside})
\]
\[
\text{plot}(X, \text{col}=\text{rgb}(1-\text{inside},0,0+\text{inside}))
\]
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 \([0,1] \times [0,1]\), instead of \([-5,0] \times [10,15]\) as usual. It has 3 global minima: \(x_1 = (0.9616520, 0.15)\); \(x_2 = (0.1238946, 0.8166644)\); \(x_3 = (0.5427730, 0.15)\).

**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 \([0,1] \times [0,1]\), instead of \([-5,0] \times [10,15]\) as usual. It has 3 global minima: \(x_1 = (0.9616520, 0.15)\); \(x_2 = (0.1238946, 0.8166644)\); \(x_3 = (0.5427730, 0.15)\).

**Usage**

\[
\text{branin}(x)
\]

**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 \(x\).

d3n.factor

**Description**

Generalize expand.grid() for multi-columns data. Build all combinations of lines from \(X_1\) and \(X_2\). Each line may hold multiple columns.

**Usage**

\[
\text{combn.design}(X_1, X_2)
\]

**Arguments**

- \(X_1\) variable values, possibly with many columns
- \(X_2\) variable values, possibly with many columns
Plot a contour view of a prediction model or function, including design points if available.

### Usage

```r
## S3 method for class 'function'
contourview(
  fun,
  vectorized = FALSE,
  dim = NULL,
  center = NULL,
  axis = NULL,
  npoints = 20,
  nlevels = 10,
  col_surf = "blue",
  filled = FALSE,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'matrix'
contourview(
  X,
  y,
  sdy = NULL,
  center = NULL,
  axis = NULL,
  col_points = "red",
  bg_blend = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  title = NULL,
  add = FALSE,
  ...
)
```
## S3 method for class 'km'
contourview(
  km_model,
  type = "UK",
  center = NULL,
  axis = NULL,
  npoints = 20,
  nlevels = 10,
  col_points = "red",
  col_surf = "blue",
  filled = FALSE,
  bg_blend = 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 = 20,
  nlevels = 10,
  col_points = "red",
  col_surf = "blue",
  filled = FALSE,
  bg_blend = 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 = 20,
contourview

nlevels = 10,
col_points = "red",
col_surf = "blue",
filled = FALSE,
bg_blend = 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 = 20,
  nlevels = 10,
  col_points = "red",
  col_surf = "blue",
  filled = FALSE,
  bg_blend = 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 = 20,
  nlevels = 10,
  col_points = "red",
  col_surf = "blue",
  filled = FALSE,
  bg_blend = 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 = 20,  
nlevels = 10,  
col_points = "red",  
col_surf = "blue",  
bg_blend = 1,  
filled = FALSE,  
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.
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_surf 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.
Xlab    an optional list of string to overload names for X.
contourview.function

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.
sdy optional array of output standard error.
col_points color of points.
bg_blend an optional factor of alpha (color channel) blending used to plot design points outside from this section.
km_model an object of class "km".
type the kriging type to use for model prediction.
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.
Vectorize.function to wrap as vectorized a non-vectorized function.
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.

Examples

```r
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) <- names(model$coefficients[-1])
    p = predict.lm(model, newdata=x, se.fit=TRUE)
    list(mean=p$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)

contourview(X, y)

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, nlevels=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
```
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))
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(-0.5,ncol=2,nrow=1),p=matrix(c(0,0,1,0,0),ncol=2))
is_in.p(x=matrix(0.25,ncol=2,nrow=1),p=matrix(c(0,0,1,0,0,0,0,1),ncol=2))
is_in.p(x=matrix(-0.5,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(0.25,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 evaluations.

Usage

Memoize.function(fun)

Arguments

fun  
function to memoize
Value

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

Examples

```r
f=function(n) rnorm(n);
F=Memoize.function(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**

```r
mesh_exsets(
  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'
- `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` 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),
# 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$p[tri,],col=rgb(.4,.4,.4,.4)))
  if (requireNamespace("rgl")) {
    e = mesh_exsets(function(x) (0.5+x[1])^2+(0.5+x[2])^2+(0.5+x[3])^2,
      threshold = .25,sign=1, mesh="unif",
      intervals=matrix(c(-1,1,-1,1,-1,1),nrow=2),
      maxerror_f=1E-2,tol=1E-2) # for faster testing
    rgl::plot3d(e$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_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,
  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
- **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

```r
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+(0.5+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

```r
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 vector of distance (same number of columns than x)

Value

minimal distance

Examples

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

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(par, 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

```r
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")
optim(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
optim(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

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

Arguments

- `mesh`: 2-dimensional mesh to draw
- `color`: color of the mesh
- `...`: optional arguments passed to plot function

Examples

```r
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, engine3d = NULL, color = "black", ...)

Arguments

mesh 3-dimensional mesh to draw
engine3d 3d framework to use: 'rgl' if installed or 'scatterplot3d' (default)
color color of the mesh
... 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="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="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 = "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**

```r
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**

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

**Arguments**

- **mesh**

**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**

```r
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**

```r
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)
One Dimensional Multiple Roots (Zero) Finding

Description

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

Usage

```r
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

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

```r
f=function(x)exp(x)-1;
```

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

```r
## S3 method for class "function"
sectionview(
  fun,
  vectorized = FALSE,
  dim = NULL,
  center = NULL,
  axis = NULL,
  npoints = 100,
  col_surf = "blue",
  conf_lev = c(0.5, 0.8, 0.9, 0.95, 0.99),
  conf_blend = NULL,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  ...
)

## S3 method for class 'matrix'
sectionview(
  X,
  y,
  sdy = NULL,
  center = NULL,
  axis = NULL,
  npoints = 100,
  col_points = "red",
  conf_lev = c(0.5, 0.8, 0.9, 0.95, 0.99),
  conf_blend = NULL,
  bg_blend = 5,
  ...)```
sectionview.function

```r
mfrow = NULL,
Xlab = NULL,
ylab = NULL,
Xlim = NULL,
ylim = NULL,
title = NULL,
add = FALSE,
```

```r
## S3 method for class 'km'
sectionview(
  km_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,
  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 = 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,
  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 = 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,
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 = 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,
Xlab = NULL,
ylab = NULL,
Xlim = NULL,
ylim = NULL,
title = NULL,
add = FALSE,
...)

## S3 method for class 'glm'
sectionview.function

sectionview(
    glm_model, 
    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, 
    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 = 100, 
    col_points = "red", 
    col_surf = "blue", 
    bg_blend = 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 mean and
            standard error: list(mean=...,se=...).
vectorized  is fun vectorized?
dim         input variables dimension of the model or function.
sectionview.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(2, 2).
npoints an optional number of points to discretize plot of response surface and uncertainties.
col_surf color for the surface.
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.
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.
sdy optional array of output standard error.
col_points color of points.
bg_blend an optional factor of alpha (color channel) blending used to plot design points outside from this section.
km_model an object of class "km".
type the kriging type to use for model prediction.
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.
see Also

`sectionview.function` for a section plot, and `sectionview3d.function` for a 2D section plot. `Vectorize.function` to wrap as vectorized a non-vectorized function.

`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.

Examples

```r
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) <- names(model$coefficients[-1])
  p = predict.lm(model, newdata=x, se.fit=TRUE)
  list(mean=p$fit, se=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))

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")
```
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

```r
sectionview(branin, center= c(.5,.5), col='black')
```

```r
## 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)
```
Usage

## S3 method for class 'function'
sectionview3d(
  fun,
  vectorized = FALSE,
  dim = NULL,
  center = NULL,
  axis = NULL,
  npoints = 20,
  col_surf = "blue",
  conf_lev = c(0.95),
  conf_blend = NULL,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'matrix'
sectionview3d(
  X,
  y,
  sdy = NULL,
  center = NULL,
  axis = NULL,
  col_points = "red",
  conf_lev = c(0.95),
  conf_blend = NULL,
  bg_blend = 1,
  mfrow = NULL,
  Xlab = NULL,
  ylab = NULL,
  Xlim = NULL,
  ylim = NULL,
  title = NULL,
  add = FALSE,
  engine3d = NULL,
  ...
)

## S3 method for class 'km'
sectionview3d(
  km_model,
### sectionview3d function

```r
sectionview3d(type = "UK",
               center = NULL,
               axis = NULL,
               npoints = 20,
               col_points = "red",
               col_surf = "blue",
               conf_lev = c(0.95),
               conf_blend = NULL,
               bg_blend = 1,
               mfrow = NULL,
               Xlab = NULL,
               ylab = NULL,
               Xlim = NULL,
               ylim = NULL,
               title = NULL,
               add = FALSE,
               engine3d = NULL,
               ...
)
```

## S3 method for class 'Kriging'

```r
sectionview3d(Kriging_model,
               center = NULL,
               axis = NULL,
               npoints = 20,
               col_points = "red",
               col_surf = "blue",
               conf_lev = c(0.95),
               conf_blend = NULL,
               bg_blend = 1,
               mfrow = NULL,
               Xlab = NULL,
               ylab = NULL,
               Xlim = NULL,
               ylim = NULL,
               title = NULL,
               add = FALSE,
               engine3d = NULL,
               ...
)
```

## S3 method for class 'NuggetKriging'

```r
sectionview3d(NuggetKriging_model,
               center = NULL,
               axis = NULL,
               npoints = 20,
               npoints = 20,
               d
```
sectionview3d.function

sectionview3d(            
  NoiseKriging_model,        
  center = NULL,              
  axis = NULL,               
  npoints = 20,              
  col_points = "red",        
  col_surf = "blue",         
  conf_lev = c(0.95),        
  conf_blend = NULL,         
  bg_blend = 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 = 20,              
  col_points = "red",        
  col_surf = "blue",         
  conf_lev = c(0.95),        
  conf_blend = NULL,         
  bg_blend = 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 = 20,              
  col_points = "red",        
  col_surf = "blue",         
  conf_lev = c(0.95),        
  conf_blend = NULL,         
  bg_blend = 1,              
  mfrow = NULL,              
  Xlab = NULL,               
  ylab = NULL,               
  Xlim = NULL,               
  ylim = NULL,               
  title = NULL,              
  add = FALSE,               
  engine3d = NULL,           
  ...
)
sectionview3d.function

```r
bg_blend = 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 = 20,
col_points = "red",
col_surf = "blue",
bg_blend = 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 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_surf**

- color for the surface.
sectionview3d.function

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.
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.
sdy optional array of output standard error.
col_points color of points.
bg_blend an optional factor of alpha (color channel) blending used to plot design points outside from this section.
km_model an object of class "km".
type the kriging type to use for model prediction.
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. `Vectorize.function` to wrap as vectorized a non-vectorized function.

`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.

Examples

```r
x1 <- rnorm(15)
x2 <- rnorm(15)
y <- x1 + x2 + rnorm(15)
DiceView:::open3d(); DiceView:::plot3d(x1,x2,y)

model <- lm(y ~ x1 + x2)

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) <- names(model$coefficients[-1])
  p = predict.lm(model, newdata=x, se.fit=TRUE)
  list(mean=p$fitted, 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)

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
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, ...)
Arguments

fun 'dim'-dimensional function to Vectorize

Arguments

dim dimension of input arguments of fun

Value

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));
Index

Apply.function, 2
are_in.mesh, 3
branin, 4

combn.design, 4
contourview(contourview.function), 5
contourview(function, function-method (contourview.function), 5
contourview, glm, glm-method (contourview.function), 5
contourview, km, km-method (contourview.function), 5
contourview,Kriging,Kriging-method (contourview.function), 5
contourview,list,list-method (contourview.function), 5
contourview, matrix, matrix-method (contourview.function), 5
contourview, NoiseKriging, NoiseKriging-method (contourview.function), 5
contourview, NuggetKriging, NuggetKriging-method (contourview.function), 5
contourview.function, 5
contourview, glm (contourview.function), 5
contourview, km (contourview.function), 5
contourview, Kriging (contourview.function), 5
contourview, list (contourview.function), 5
contourview, matrix (contourview.function), 5
contourview, NoiseKriging (contourview.function), 5
contourview, NuggetKriging (contourview.function), 5

is_in.mesh, 12
is_in.p, 13

Memoize.function, 13
mesh_exsets, 14
mesh_roots, 15
min_dist, 16

optims, 17
plot2d_mesh, 18
plot3d_mesh, 19
plot_mesh, 19
points_in.mesh, 20
points_out.mesh, 20

root, 21
roots, 23

sectionview(sectionview.function), 24
sectionview, function, function-method (sectionview.function), 24
sectionview, glm, glm-method (sectionview.function), 24
sectionview, km, km-method (sectionview.function), 24
sectionview,Kriging,Kriging-method (sectionview.function), 24
sectionview,list,list-method (sectionview.function), 24
sectionview, matrix, matrix-method (sectionview.function), 24
sectionview, NoiseKriging, NoiseKriging-method (sectionview.function), 24
sectionview, NuggetKriging, NuggetKriging-method (sectionview.function), 24
sectionview.function, 9, 24, 29, 37
sectionview, glm, 10, 29, 37
sectionview, glm (sectionview.function), 24
sectionview, km, 9, 29, 37
sectionview, km (sectionview.function), 24
sectionview.Kriging, 9, 29, 37
sectionview.Kriging
(sectionview.function), 24
sectionview.list
(sectionview.function), 24
sectionview.matrix, 9, 29, 37
sectionview.matrix
(sectionview.function), 24
sectionview.NoiseKriging, 10, 29, 37
sectionview.NoiseKriging
(sectionview.function), 24
sectionview.NuggetKriging, 9, 29, 37
sectionview.NuggetKriging
(sectionview.function), 24
sectionview3d(sectionview3d.function), 31
sectionview3d.function, function-method
(sectionview3d.function), 31
sectionview3d.glm, glm-method
(sectionview3d.function), 31
sectionview3d.km, km-method
(sectionview3d.function), 31
sectionview3d.Kriging, Kriging-method
(sectionview3d.function), 31
sectionview3d.list, list-method
(sectionview3d.function), 31
sectionview3d.matrix, matrix-method
(sectionview3d.function), 31
sectionview3d.NoiseKriging, NoiseKriging-method
(sectionview3d.function), 31
sectionview3d.NuggetKriging, NuggetKriging-method
(sectionview3d.function), 31
sectionview3d.function, 9, 29, 31, 37
sectionview3d.glm, 10, 29, 37
sectionview3d.glm
(sectionview3d.function), 31
sectionview3d.km, 9, 29, 37
sectionview3d.km
(sectionview3d.function), 31
sectionview3d.Kriging, 9, 29, 37
sectionview3d.Kriging
(sectionview3d.function), 31
sectionview3d.list
(sectionview3d.function), 31
sectionview3d.matrix, 9, 29, 37
sectionview3d.matrix
(sectionview3d.function), 31
sectionview3d.NoiseKriging, 10, 29, 37