Apply.function

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.
are.in.mesh

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))
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: \(x_1 = \text{c}(0.9616520, 0.15)\); \(x_2 = \text{c}(0.1238946, 0.8166644)\); \(x_3 = \text{c}(0.5427730, 0.15)\).

**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

```r
combn.design(matrix(c(10,20),ncol=1),matrix(c(1,2,3,4,5,6),ncol=2))
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

```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,
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(...)
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)
}
```
contourview.function

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

\[
\text{is\_in\_p}(x, p, h = \text{NULL})
\]

Arguments

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

Examples

\[
\begin{align*}
\text{is\_in\_p}(x=-0.5, p=\text{matrix}(c(0, 1), ncol=1)) \\
\text{is\_in\_p}(x=0.5, p=\text{matrix}(c(0, 1), ncol=1)) \\
\text{is\_in\_p}(x=\text{matrix}(-.5, ncol=2, nrow=1), p=\text{matrix}(c(0, 0, 1, 1, 0, 0), ncol=2)) \\
\text{is\_in\_p}(x=\text{matrix}(.25, ncol=2, nrow=1), p=\text{matrix}(c(0, 0, 1, 1, 0, 0), ncol=2)) \\
\text{is\_in\_p}(x=\text{matrix}(-.5, ncol=3, nrow=1), p=\text{matrix}(c(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1), ncol=3, byrow = \text{TRUE})) \\
\text{is\_in\_p}(x=\text{matrix}(.25, ncol=3, nrow=1), p=\text{matrix}(c(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1), ncol=3, byrow = \text{TRUE}))
\end{align*}
\]

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

\[
\text{Memoize.function}(\text{fun})
\]

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_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 = "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 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

```r
# 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="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

```r
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, 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)))
```
**plot2d_mesh**

**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**

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

---

**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", ...)

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')
}
}

**points_in.mesh**

**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)))
```

**Description**

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

**Usage**

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

**Arguments**

- **mesh**: mesh (list(p,tri,...) from geometry)

**Value**

points coordinates inside the mesh triangulation

---

**points_out.mesh**

**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 = 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) 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.
sectionview.function

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.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,
  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,
mfrow = NULL,
Xlab = NULL,
ylab = NULL,
Xlim = NULL,
ylim = NULL,
title = NULL,
add = FALSE,
...)

## 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,
sectionview.function

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(
  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.

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.

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.

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

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)
}
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

```r
## 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, 
  ...  
)
```

```r
## 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, 
)```
sectionview3d.function

mfrow = NULL,
Xlab = NULL,
ylab = NULL,
Xlim = NULL,
ylim = NULL,
title = NULL,
add = FALSE,
engine3d = NULL,
...

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

    engine3d = NULL,
    ...
)

## S3 method for class 'glm'
sectionview3d3d(
    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,
)

## S3 method for class 'list'
sectionview3d3d(
    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.
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. `sectionview3d.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
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 <- rlibkriging::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

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
dim dimension of input arguments of fun
... 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));
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
plot2d_mesh, 17
plot3d_mesh, 18
plot_mesh, 18
points_in.mesh, 19
points_out.mesh, 19
root, 20
roots, 21
sectionview(sectionview.function), 22
sectionview, function, function-method (sectionview.function), 22
sectionview, glm, glm-method (sectionview.function), 22
sectionview, km, km-method (sectionview.function), 22
sectionview, Kriging, Kriging-method (sectionview.function), 22
sectionview, list, list-method (sectionview.function), 22
sectionview, matrix, matrix-method (sectionview.function), 22
sectionview, NoiseKriging, NoiseKriging-method (sectionview.function), 22
sectionview, NuggetKriging, NuggetKriging-method (sectionview.function), 22
sectionview.function, 9, 22, 27, 35
sectionview.glm, 10, 27, 35
sectionview.glm (sectionview.function), 22
sectionview.km, 9, 27, 35
sectionview.km (sectionview.function), 22
sectionview.Kriging, 9, 27, 35
INDEX

sectionview.Kriging (sectionview.function), 22
sectionview.list (sectionview.function), 22
sectionview.matrix, 9, 27, 35
sectionview.matrix (sectionview.function), 22
sectionview.NoiseKriging, 10, 27, 35
sectionview.NoiseKriging (sectionview.function), 22
sectionview.NuggetKriging, 9, 27, 35
sectionview.NuggetKriging (sectionview.function), 22
sectionview3d (sectionview3d.function), 30
sectionview3d, function, function-method (sectionview3d.function), 30
sectionview3d, glm, glm-method (sectionview3d.function), 30
sectionview3d, km, km-method (sectionview3d.function), 30
sectionview3d, Kriging, Kriging-method (sectionview3d.function), 30
sectionview3d, list, list-method (sectionview3d.function), 30
sectionview3d, matrix, matrix-method (sectionview3d.function), 30
sectionview3d, NoiseKriging, NoiseKriging-method (sectionview3d.function), 30
sectionview3d, NuggetKriging, NuggetKriging-method (sectionview3d.function), 30
sectionview3d.function, 9, 27, 30, 35
sectionview3d.glm, 10, 27, 35
sectionview3d.glm (sectionview3d.function), 30
sectionview3d.km, 9, 27, 35
sectionview3d.km (sectionview3d.function), 30
sectionview3d.Kriging, 9, 27, 35
sectionview3d.Kriging (sectionview3d.function), 30
sectionview3d.list (sectionview3d.function), 30
sectionview3d.matrix, 9, 27, 35
sectionview3d.matrix (sectionview3d.function), 30
sectionview3d.NoiseKriging, 10, 27, 35
sectionview3d.NoiseKriging (sectionview3d.function), 30
Vectorize.function, 9, 27, 35, 37