Package ‘KBE’

October 27, 2021

Type Package

Title Known Boundary Emulation

Version 0.1.0

Author Samuel E. Jackson

Maintainer Samuel E. Jackson <samuel.e.jackson@durham.ac.uk>

Description Package for reproducing and exploring the Known Boundary Emulation examples from the article ``Efficient Emulation of Computer Models Utilising Multiple Known Boundaries of Differing Dimension".

Imports pdist, viridis

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

R topics documented:

`Article_Plots` *Quick Generation of 3D Example Plots.*

Description

Quick Generation of 3D Example Plots.

Usage

```
Article_Plots(
  f,
  ranges,
  K_d,
  L_d = NA,
  M_d = NA,
  xK_d,
  xL_d = NA,
  xM_d = NA,
  fixed_dimension,
  fixed_value,
  theta = theta,
  s2 = s2,
  zlim_f = "assessed",
  zlim_var = "assessed",
  main_cube = list("", "", "")
)
```

Arguments

`f`	a toy function for which plots will be created.
`ranges`	the ranges for the input parameters of the toy functions, given as a matrix.
`K_d`	variable indices to be fixed for boundary K.
`L_d`	variable indices to be fixed for boundary L.
`M_d`	variable indices to be fixed for boundary M.
`xK_d`	values at which those variables are fixed to.
`xL_d`	values at which those variables are fixed to.
`xM_d`	values at which those variables are fixed to.
`fixed_dimension`	index of the variable that will be kept fixed for the plots, given as a vector of length 3.
`fixed_value`	fixed value in the remaining dimension, given as a vector of length 3.
`theta`	correlation length parameters.
`s2`	scalar variance parameter.
`zlim_f`	plotting range for the z-values (model and mean prediction)
`zlim_var`	plotting range for the variance predictions.
`main_cube`	title for the cube plot, given as a list of length 3.

Value

nothing is returned. Plots are produced.

Examples

```
# Specify the toy function - requires a 3D input vector.
f <- function( x ){

 sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}

# Specify ranges for the function parameters x1, x2, and x3.
ranges <- matrix(c( -2*pi, 2*pi,
                    -pi/4, pi/4,
                    -2*pi, 2*pi), ncol = 2, byrow = TRUE)

# Specify the correlation length parameters and variance parameter for the example.
theta <- c( pi, pi/8, pi )
s2 <- 2

# Specify the ranges for the function/mean and variance plots.
zlim_f <- c(-2.5, 2.5)
zlim_var <- c(0, 2)

# Specify the boundary.
K_d <- c( 2, 3 )
xK_d <- c( 0, 0 )

# Fixed dimensions.
fixed_dimension <- c( 2, 2, 1 )
fixed_value <- c( 0, -pi/8, -pi )

# Set labels.
quotes <- list( bquote( x[2] == 0 ), bquote( x[2] == -pi/8 ), bquote( x[1] == -pi ) )

# Run the function.
Article_Plots( f = f,
               ranges = ranges,
               K_d = K_d,
               xK_d = xK_d,
               fixed_dimension = fixed_dimension,
               fixed_value = fixed_value,
               theta = theta,
               s2 = s2,
               zlim_f = zlim_f,
               zlim_var = zlim_var,
               main_cube = quotes )
```

AVEM *Array Vector-Element Multiplication*

Description

Multiply each matrix/column in a 3D/2D array by the corresponding element of a vector.

Usage

```
AVEM(A, v)
```

Arguments

A	two or three-dimensional array.
v	vector

Value

array resulting from multiplying the matrices/columns in A by the corresponding elements in v.

Examples

```
A <- array( 1:24, dim = c( 2,3,4 ) )
b = 1:4
AVEM( A, b )
```

BLA_1B *Bayes Linear Adjustment by a Single Known Boundary*

Description

Perform a Bayes linear adjustment utilising knowledge of function behaviour along a single boundary in the input space.

Usage

```
BLA_1B(x, K_d, xK = NA, xK_d = NA, fxK, E_fx = 0, E_fxK = 0, theta, s2)
```

Arguments

x	points at which we want to update
K_d	the dimensions which, when fixed at certain values, result in known boundaries.
xK	the projection of x onto known boundary K
xK_d	values the dimensions K must take for the function to be known
fxK	function evaluated at x projected onto the boundary K.
E_fx	prior expectation for the function f(x)
E_fxK	prior expectation for f(x^K)
theta	vector of correlation length parameter values
s2	scalar variance parameter value.

Value

EB_fx	Expected value of f(x) adjusted by knowledge of function behaviour along K.
VarB_fx	Variance of f(x) adjusted by knowledge of function behaviour along K.
CovB_fx	Covariance of f(x) adjusted by knowledge of function behaviour along K.

Examples

```
# Toy function
f <- function( x ){

sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}
x <- matrix( runif( 12 ), ncol = 3 )
K_d = 2
xK_d = 0
fxK <- f_boundary( x = x, K_d = K_d, xK_d = xK_d, f = f )
theta <- c( pi, pi/8, pi )
s2 <- 2
BA <- BLA_1B( x = x, K_d = K_d, xK_d = xK_d, fxK = fxK, theta = theta, s2 = s2 )
```

BLA_2parB *Bayes Linear Adjustment by 2 Parallel Known Boundaries*

Description

Perform a Bayes linear adjustment utilising knowledge of function behaviour along two parallel known boundaries in the input space.

Usage

```
BLA_2parB(
  x,
  K_d,
  L_d = 0,
  xK = NA,
  xL = NA,
  xLK = NA,
  xK_d = NA,
  xL_d = NA,
  fxK,
  fxL,
  fxLK,
  E_fx = 0,
  E_fxK = 0,
  E_fxL = 0,
  E_fxLK = 0,
  theta,
  s2
)
```

Arguments

x	points at which we want to update
K_d	the dimensions which, when fixed at certain values, result in known boundary K.

L_d	the dimensions which, when fixed at certain values, result in known boundary L.
xK	the projection of x onto known boundary K
xL	the projection of x onto known boundary L
xLK	the projection of x first onto known boundary L and then known boundary K.
xK_d	values the dimensions K must take for the function to be known
xL_d	values the dimensions L must take for the function to be known
fxK	function evaluated at x projected onto the boundary K.
fxL	function evaluated at x projected onto the boundary L.
fxLK	function evaluated at xLK.
E_fx	prior expectation for the function f(x)
E_fxK	prior expectation for f(x^K)
E_fxL	prior expectation for f(x^L)
E_fxLK	prior expectation for f(x^LK)
theta	vector of correlation length parameter values.
s2	scalar variance parameter value.

Value

EB_fx	Expected value of f(x) adjusted by knowledge of function behaviour along K and L.
VarB_fx	Variance of f(x) adjusted by knowledge of function behaviour along K and L.
CovB_fx	Covariance of f(x) adjusted by knowledge of function behaviour along K and L.

Examples

```
# Toy function
f <- function( x ){

sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}
x <- matrix( runif( 12 ), ncol = 3 )
K_d = 2
L_d = c(2,3)
xK_d = 0
xL_d = c(1,1)
# If we are in a parallel setting, then xLK (projection of x first onto L and then K)
# is given as follows:
xLK_d <- xL_d
xLK_d[1:length(xK_d)] <- xK_d
# And LK_d (fixed values of coordinates for projections first onto L and then K)
# is just given by L_d.
LK_d <- L_d
fxK <- f_boundary( x = x, K_d = K_d, xK_d = xK_d, f = f )
fxL <- f_boundary( x = x, K_d = L_d, xK_d = xL_d, f = f )
fxLK <- f_boundary( x = x, K_d = LK_d, xK_d = xLK_d, f = f )
theta <- c( pi, pi/8, pi )
s2 <- 2
BA <- BLA_2parB( x = x, K_d = K_d, L_d = L_d, xK_d = xK_d, xL_d = xL_d,
                  fxK = fxK, fxL = fxL, fxLK = fxLK, theta = theta, s2 = s2 )
```

BLA_2perpB *Bayes Linear Adjustment by 2 Perpendicular Known Boundaries*

Description

Perform a Bayes linear adjustment utilising knowledge of function behaviour along two perpendicular known boundaries in the input space.

Usage

```
BLA_2perpB(
  x,
  K_d,
  L_d,
  xK = NA,
  xL = NA,
  xLK = NA,
  xK_d = NA,
  xL_d = NA,
  fxK,
  fxL,
  fxLK,
  E_fx = 0,
  E_fxK = 0,
  E_fxL = 0,
  E_fxLK = 0,
  theta,
  s2
)
```

Arguments

`x`	points at which we want to update
`K_d`	the dimensions which, when fixed at certain values, result in known boundary K.
`L_d`	the dimensions which, when fixed at certain values, result in known boundary L.
`xK`	the projection of x onto known boundary K
`xL`	the projection of x onto known boundary L
`xLK`	the projection of x onto the intersection of known boundaries K and L.
`xK_d`	values the dimensions K must take for the function to be known
`xL_d`	values the dimensions L must take for the function to be known
`fxK`	function evaluated at x projected onto the boundary K.
`fxL`	function evaluated at x projected onto the boundary L.
`fxLK`	function evaluated at x projected onto the intersection of boundaries K and L.
`E_fx`	prior expectation for the function f(x)
`E_fxK`	prior expectation for f(x^K)

E_fxL	prior expectation for f(x^L)
E_fxLK	prior expectation for f(x^LK)
theta	vector of correlation length parameter values.
s2	scalar variance parameter value.

Value

EB_fx	Expected value of f(x) adjusted by knowledge of function behaviour along K and L.
VarB_fx	Variance of f(x) adjusted by knowledge of function behaviour along K and L.
CovB_fx	Covariance of f(x) adjusted by knowledge of function behaviour along K and L.

Examples

```
# Toy function
f <- function( x ){

sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}
x <- matrix( runif( 12 ), ncol = 3 )
K_d = 2
L_d = 1
xK_d = 0
xL_d = 0
fxK <- f_boundary( x = x, K_d = K_d, xK_d = xK_d, f = f )
fxL <- f_boundary( x = x, K_d = L_d, xK_d = xL_d, f = f )
fxLK <- f_boundary( x = x, K_d = c(K_d, L_d), xK_d = c(xK_d, xL_d), f = f )
theta <- c( pi, pi/8, pi )
s2 <- 2
BA <- BLA_2perpB( x = x, K_d = K_d, L_d = L_d, xK_d = xK_d, xL_d = xL_d,
                  fxK = fxK, fxL = fxL, fxLK = fxLK, theta = theta, s2 = s2 )
```

BLA_3B *Bayes Linear Adjustment by 3 Known Boundaries*

Description

Perform a Bayes linear adjustment utilising knowledge of function behaviour along three known boundaries in the input space. In this case boundaries K and L should be parallel to each other, and M should be perpendicular to K and L.

Usage

```
BLA_3B(
  x,
  K_d,
  L_d,
  M_d,
  xK = NA,
  xL = NA,
```

```
    xM = NA,
    xLK = NA,
    xMK = NA,
    xML = NA,
    xMLK = NA,
    xK_d = NA,
    xL_d = NA,
    xM_d = NA,
    fxK,
    fxL,
    fxM,
    fxLK,
    fxMK,
    fxML,
    fxMLK,
    E_fx = 0,
    E_fxK = 0,
    E_fxL = 0,
    E_fxM = 0,
    E_fxLK = 0,
    E_fxMK = 0,
    E_fxML = 0,
    E_fxMLK = 0,
    theta,
    s2
)
```

Arguments

`x`	points at which we want to update
`K_d`	the dimensions which, when fixed at certain values, result in known boundary K.
`L_d`	the dimensions which, when fixed at certain values, result in known boundary L.
`M_d`	the dimensions which, when fixed at certain values, result in known boundary M.
`xK`	the projection of x onto known boundary K
`xL`	the projection of x onto known boundary L
`xM`	the projection of x onto known boundary M
`xLK`	the projection of x first onto known boundary L and then known boundary K.
`xMK`	the projection of x onto the intersection of known boundaries K and M.
`xML`	the projection of x onto the intersection of known boundaries L and M.
`xMLK`	the projection of x onto the intersection of M and that obtained by projecting first onto L and then onto K.
`xK_d`	values the dimensions K must take for the function to be known
`xL_d`	values the dimensions L must take for the function to be known
`xM_d`	values the dimensions M must take for the function to be known
`fxK`	function evaluated at x projected onto the boundary K.

fxL	function evaluated at x projected onto the boundary L.
fxM	function evaluated at x projected onto the boundary M.
fxLK	function evaluated at xLK.
fxMK	function evaluated at xMK.
fxML	function evaluated at xML.
fxMLK	function evaluated at xMLK.
E_fx	prior expectation for the function f(x)
E_fxK	prior expectation for f(x^K)
E_fxL	prior expectation for f(x^L)
E_fxM	prior expectation for f(x^M)
E_fxLK	prior expectation for f(x^LK)
E_fxMK	prior expectation for f(x^MK)
E_fxML	prior expectation for f(x^ML)
E_fxMLK	prior expectation for f(x^MLK)
theta	vector of correlation length parameter values.
s2	scalar variance parameter value.

Value

EB_fx	Expected value of f(x) adjusted by knowledge of function behaviour along K, L and M.
VarB_fx	Variance of f(x) adjusted by knowledge of function behaviour along K, L and M.
CovB_fx	Covariance of f(x) adjusted by knowledge of function behaviour along K, L and M.

Examples

```
# Toy function
f <- function( x ){

sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}
x <- matrix( runif( 12 ), ncol = 3 )
K_d = 2
L_d = c(2,3)
M_d = 1
xK_d = 0
xL_d = c(1,1)
xM_d = 0
#' # If we are in a parallel setting, then xLK (projection of x first onto L and then K)
# is given as follows:
xLK_d <- xL_d
xLK_d[1:length(xK_d)] <- xK_d
# And LK_d (fixed values of coordinates for projections first onto L and then K)
# is just given by L_d.
LK_d <- L_d
fxK <- f_boundary( x = x, K_d = K_d, xK_d = xK_d, f = f )
```

```
fxL <- f_boundary( x = x, K_d = L_d, xK_d = xL_d, f = f )
fxM <- f_boundary( x = x, K_d = M_d, xK_d = xM_d, f = f )
fxLK <- f_boundary( x = x, K_d = LK_d, xK_d = xLK_d, f = f )
fxMK <- f_boundary( x = x, K_d = c(K_d, M_d), xK_d = c(xK_d, xM_d), f = f )
fxML <- f_boundary( x = x, K_d = c(L_d, M_d), xK_d = c(xL_d, xM_d), f = f )
fxMLK <- f_boundary( x = x, K_d = c(LK_d, M_d), xK_d = c(xLK_d, xM_d), f = f )
theta <- c( pi, pi/8, pi )
s2 <- 2
BA <- BLA_3B( x = x, K_d = K_d, L_d = L_d, M_d = M_d,
              xK_d = xK_d, xL_d = xL_d, xM_d = xM_d,
              fxK = fxK, fxL = fxL, fxM = fxM,
              fxLK = fxLK, fxMK = fxMK, fxML = fxML, fxMLK = fxMLK,
              theta = theta, s2 = s2 )
```

BLA_3perpB *Bayes Linear Adjustment by 3 Perpendicular Known Boundaries*

Description

Perform a Bayes linear adjustment utilising knowledge of function behaviour along three perpendicular known boundaries in the input space.

Usage

```
BLA_3perpB(
  x,
  K_d,
  L_d,
  M_d,
  xK = NA,
  xL = NA,
  xM = NA,
  xLK = NA,
  xMK = NA,
  xML = NA,
  xMLK = NA,
  xK_d = NA,
  xL_d = NA,
  xM_d = NA,
  fxK,
  fxL,
  fxM,
  fxLK,
  fxMK,
  fxML,
  fxMLK,
  E_fx = 0,
  E_fxK = 0,
  E_fxL = 0,
  E_fxM = 0,
  E_fxLK = 0,
  E_fxMK = 0,
```

```
  E_fxML = 0,
  E_fxMLK = 0,
  theta,
  s2
)
```

Arguments

x	points at which we want to update
K_d	the dimensions which, when fixed at certain values, result in known boundary K.
L_d	the dimensions which, when fixed at certain values, result in known boundary L.
M_d	the dimensions which, when fixed at certain values, result in known boundary M.
xK	the projection of x onto known boundary K
xL	the projection of x onto known boundary L
xM	the projection of x onto known boundary M
xLK	the projection of x onto the intersection of known boundaries K and L.
xMK	the projection of x onto the intersection of known boundaries K and M.
xML	the projection of x onto the intersection of known boundaries L and M.
xMLK	the projection of x onto the intersection of known boundaries K, L and M.
xK_d	values the dimensions K must take for the function to be known
xL_d	values the dimensions L must take for the function to be known
xM_d	values the dimensions M must take for the function to be known
fxK	function evaluated at x projected onto the boundary K.
fxL	function evaluated at x projected onto the boundary L.
fxM	function evaluated at x projected onto the boundary M.
fxLK	function evaluated at x projected onto the intersection of boundaries K and L.
fxMK	function evaluated at x projected onto the intersection of boundaries K and M.
fxML	function evaluated at x projected onto the intersection of boundaries L and M.
fxMLK	function evaluated at x projected onto the intersection of boundaries K, L and M.
E_fx	prior expectation for the function f(x)
E_fxK	prior expectation for f(x^K)
E_fxL	prior expectation for f(x^L)
E_fxM	prior expectation for f(x^M)
E_fxLK	prior expectation for f(x^LK)
E_fxMK	prior expectation for f(x^MK)
E_fxML	prior expectation for f(x^ML)
E_fxMLK	prior expectation for f(x^MLK)
theta	vector of correlation length parameter values.
s2	scalar variance parameter value.

Value

EB_fx	Expected value of f(x) adjusted by knowledge of function behaviour along K, L and M.
VarB_fx	Variance of f(x) adjusted by knowledge of function behaviour along K, L and M.
CovB_fx	Covariance of f(x) adjusted by knowledge of function behaviour along K, L and M.

Examples

```
# Toy function
f <- function( x ){

sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}
x <- matrix( runif( 12 ), ncol = 3 )
K_d = 2
L_d = 1
M_d = 3
xK_d = 0
xL_d = 0
xM_d = 0
fxK <- f_boundary( x = x, K_d = K_d, xK_d = xK_d, f = f )
fxL <- f_boundary( x = x, K_d = L_d, xK_d = xL_d, f = f )
fxM <- f_boundary( x = x, K_d = M_d, xK_d = xM_d, f = f )
fxLK <- f_boundary( x = x, K_d = c(K_d, L_d), xK_d = c(xK_d, xL_d), f = f )
fxMK <- f_boundary( x = x, K_d = c(K_d, M_d), xK_d = c(xK_d, xM_d), f = f )
fxML <- f_boundary( x = x, K_d = c(L_d, M_d), xK_d = c(xL_d, xM_d), f = f )
fxMLK <- f_boundary( x = x, K_d = c(K_d, L_d, M_d), xK_d = c(xK_d, xL_d, xM_d), f = f )
theta <- c( pi, pi/8, pi )
s2 <- 2
BA <- BLA_3perpB( x = x, K_d = K_d, L_d = L_d, M_d = M_d,
                  xK_d = xK_d, xL_d = xL_d, xM_d = xM_d,
                  fxK = fxK, fxL = fxL, fxM = fxM,
                  fxLK = fxLK, fxMK = fxMK, fxML = fxML, fxMLK = fxMLK,
                  theta = theta, s2 = s2 )
```

---

boundary_for_plot *Boundary Coordinate Generation*

Description

A function for generating the coordinate matrices (with 2 columns, and 4 rows for 2D boundaries and 2 rows for 1D boundaries) required for the cube.

Usage

```
boundary_for_plot(fixed_dimension, fixed_value, ranges)
```

Arguments

fixed_dimension
: the dimensions which are fixed for the boundary

fixed_value
: the values for the coordinates which are fixed (of same length as fixed_dimension).

ranges
: ranges of the three variables, given as a 3 x 2 matrix.

Value

A matrix with 2 columns, and 4 rows for 2D boundaries and 2 rows for 1D boundaries This gives the coordinates of the boundaries for plotting on the cube as shown in the top row of the 3D example figures in the article.

Examples

```
ranges <- matrix(c( -2*pi, 2*pi,
                    -pi/4, pi/4,
                    -2*pi, 2*pi), ncol = 2, byrow = TRUE)
boundary_for_plot( fixed_dimension = 2,
                   fixed_value = 0,
                   ranges = ranges )
```

---

contour_plot *Generate Contour Plot*

Description

User-friendly wrapper for generating the contour plots as of the 3D example shown in the text.

Usage

```
contour_plot(x, y, z, levels, colours)
```

Arguments

x
: x coordinates.

y
: y coordinates.

z
: matrix of values to be plotted, with rows assumed to correspond to increasing values of x (from top to bottom)

levels
: levels at which z should be divided into for the contour plot.

colours
: colours for each level of z.

Value

nothing is returned. Contour plot is generated.

Examples

```
x = seq( from = 0 , to = 1, by = 0.01 )
y = seq( from = 0 , to = 1, by = 0.01 )
eg <- expand.grid( x, y )
z <- matrix( apply( eg, 1, FUN = function(x)( cos(x[1]) + tan(x[2]) )^2 ), nrow = 101 )
nlevels = 20
levels <- pretty( c(0,7), nlevels )
colours <- grDevices::colorRampPalette(
              c( "white", "cyan", "blue", "purple", "pink" ), space = "Lab" )
contour_plot( x = x, y = y, z = z, levels = levels, colours = colours )
```

---

draw_cube *Draw a Cube*

Description

Specific function for the diagrams of a cube illustrating which boundaries are known and which cross-section of the input space is being emulated.

Usage

```
draw_cube(
  lwd = 2,
  lty2 = 2,
  cex = 1.8,
  col_line_width = lwd,
  coloured_hp = list(),
  col_hp = c("green", "red", "blue", "pink"),
  density_col = rep(0.7, length(coloured_hp)),
  main = "",
  cex.main = 1,
  main.line = 1
)
```

Arguments

`lwd`	line width for the edges of the cube.
`lty2`	line type for the "hidden" edges of the cube.
`cex`	scale size for the labels.
`col_line_width`	line width for the coloured lines representing 1D boundaries.
`coloured_hp`	a list of vectors, where each vector represents the hyperplanes to be plotted:
`col_hp`	the colour of the hyperplanes given above.
`density_col`	the density of the fill of the 2D hyperplanes (note that the length of this vector
`main`	title of the cube plot.
`cex.main`	size of the title.
`main.line`	number of lines outwards from the plot edge to plot the title.

Value

nothing is returned. Cube is plotted.

Examples

```
ranges <- matrix(c( -2*pi, 2*pi,
                    -pi/4, pi/4,
                    -2*pi, 2*pi), ncol = 2, byrow = TRUE)
boundary_for_plot <- boundary_for_plot( fixed_dimension = 2,
                                        fixed_value = 0,
                                        ranges = ranges )
draw_cube( coloured_hp = list( boundary_for_plot ) )
```

`f_boundary` *Evaluate f along a boundary.*

Description

Evaluate f along a boundary.

Usage

```
f_boundary(x, K_d, xK_d, f)
```

Arguments

x	set of points, given as a matrix.
K_d	Variable indices to be fixed.
xK_d	Values at which those variables are fixed to.
f	a toy function which is to be evaluated.

Value

Value of f(x) at the projections of x projected onto boundary K_d.

Examples

```
# Toy function
f <- function( x ){

sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}
x <- matrix( runif( 12 ), ncol = 3 )
f_boundary( x = x,
            K_d = 2,
            xK_d = 0,
            f = f )
```

GaussianCF *Gaussian Correlation Function*

Description

Calculate the Gaussian correlation function between the points in (given by the rows of) two matrices.

Usage

```
GaussianCF(X, Y = X, theta, delta = 0)
```

Arguments

`X`	a vector, matrix or dataframe
`Y`	a vector, matrix or dataframe
`theta`	a vector of correlation length parameter values (one for each column of X).
`delta`	an (optional) scalar nugget parameter.

Value

Gaussian correlation function value between the rows of X and Y, given as a matrix of dimension `nrow(X)` by `nrow(Y)`.

Examples

```
X <- matrix( rnorm( 10 ), ncol = 2 )
Y <- matrix( runif( 6 ), ncol = 2 )
theta <- c( 0.5, 0.8 )
GaussianCF( X, Y, theta )
GaussianCF( as.data.frame(X), Y, theta )
```

legend_generation *Legend Generation*

Description

Generate the legend for the 3D example as shown in the article.

Usage

```
legend_generation(colours, levels)
```

Arguments

`colours`	colours for each level of z.
`levels`	levels at which z should be divided into for the contour plot.

Value

nothing is returned. Legend is plotted.

Examples

```
nlevels = 20
levels <- pretty( c(0,7), nlevels )
colours <- grDevices::colorRampPalette(
              c( "white", "cyan", "blue", "purple", "pink" ), space = "Lab" )
legend_generation( colours = colours, levels = levels )
```

`PlotGen3dEx` *3D Example Plot Generation*

Description

Generic Generation of Plots for 3D Toy Example.

Usage

```
PlotGen3dEx(
  f,
  ranges,
  K_d,
  L_d = NA,
  M_d = NA,
  xK_d,
  xL_d = NA,
  xM_d = NA,
  fixed_dimension,
  fixed_value,
  theta = rep(pi/2, 3),
  s2 = 1,
  grid_length = 50,
  lwd = 2,
  lty2 = 2,
  cex = 1.8,
  col_line_width = 3,
  zlim_f = "assessed",
  zlim_var = "assessed",
  zlim_diag = c(-4.25, 4.25),
  legend = FALSE,
  main_cube = "",
  cex.main_cube = 1.8,
  main.line = 0.2
)
```

Arguments

`f`	a toy function for which plots will be created.
`ranges`	the ranges for the input parameters of the toy functions, given as a matrix.

K_d	variable indices to be fixed for boundary K.
L_d	variable indices to be fixed for boundary L.
M_d	variable indices to be fixed for boundary M.
xK_d	values at which those variables are fixed to.
xL_d	values at which those variables are fixed to.
xM_d	values at which those variables are fixed to.
fixed_dimension	index of the variable that will be kept fixed for the plots.
fixed_value	fixed value in the remaining dimension.
theta	correlation length parameters.
s2	scalar variance parameter.
grid_length	number of grid points along each dimension with which to represent the plotted surface.
lwd	line width for cube edges.
lty2	line type for background lines
cex	relative size of plot
col_line_width	width of coloured boundary lines
zlim_f	plotting range for the z-values (model and mean prediction)
zlim_var	plotting range for the variance predictions.
zlim_diag	plotting range for the diagnostic plot.
legend	Add a legend to the side of the plot?
main_cube	title for the cube plot.
cex.main_cube	font size of the cube plot title.
main.line	line value for the title function.

Value

nothing is returned. Plots are generated.

Examples

```
# Specify the toy function - requires a 3D input vector.
f <- function( x ){

 sin( x[1] / ( exp( x[2] ) ) ) + cos( x[3] )

}

# Specify ranges for the function parameters x1, x2, and x3.
ranges <- matrix(c( -2*pi, 2*pi,
                    -pi/4, pi/4,
                    -2*pi, 2*pi), ncol = 2, byrow = TRUE)

# Specify the correlation length parameters and variance parameter for the example.
theta <- c( pi, pi/8, pi )
s2 <- 2

# Specify the ranges for the function/mean and variance plots.
```

```
zlim_f <- c(-2.5, 2.5)
zlim_var <- c(0, 2)

plot_setup()
PlotGen3dEx( f = f,
             ranges = ranges,
             K_d = c(2,3),
             L_d = c(2,3),
             M_d = 1,
             xK_d = c(0,0),
             xL_d = c(0,-pi),
             xM_d = 0,
             fixed_dimension = 2,
             fixed_value = 0,
             theta = theta,
             s2 = s2,
             zlim_f = zlim_f,
             zlim_var = zlim_var,
             main_cube = bquote( x[2] == 0 ),
             legend = FALSE )
```

---

plot_setup *Plot Setup for Article.*

Description

A simple function for setting up the plot domain to generate figures similar to those presented for the 3D example shown in the article.

Usage

```
plot_setup()
```

Value

Sets up plot domain.

Examples

```
plot_setup()
```

Scale *Scale points*

Description

Scale point from one hypercuboid domain space to another.

Usage

```
Scale(x, a = 0, b = 1, l = -1, u = 1)
```

Arguments

x	vector, matrix or dataframe of points (given by the elements or rows respectively) of points to scale.
a	vector of lower limits of the original space, one for each dimension (column of x) If all the lower limits are the same, that scalar value can be given.
b	vector of upper limits of the original space, one for each dimension (column of x) If all the upper limits are the same, that scalar value can be given.
l	vector of lower limits of the transformed space, one for each dimension (column of x) If all the lower limits are the same, that scalar value can be given.
u	vector of upper limits of the transformed space, one for each dimension (column of x) If all the upper limits are the same, that scalar value can be given.

Details

Scales the vector of matrix of points x from the hypercuboid [a,b] to [l,u].

Value

a vector or matrix of the transformed points.

Examples

```
X <- matrix( runif(15, 2, 4), ncol = 3 )
Scale( X, a = 2, b = 4 )
# Compare with:
X - 3
```

Index