Package ‘NetworkPPBLE’

August 25, 2021

Type Package

Title Partial Parallel Bayes Linear Emulation for Networks

Version 0.1.0

Author Samuel E. Jackson

Maintainer Samuel E. Jackson <s.e.jackson@soton.ac.uk>

Description Partial Parallel Bayes Linear Emulation for Networks.

Imports pdist, QuickFunc, tgp, MASS, assertthat, rSPDE

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Suggests testthat (>= 2.1.0)

RoxygenNote 7.1.1

R topics documented:

AVS *Active Variable Selection (Via First Order Linear Model)*

Description

A method for selecting the active variables of a model by fitting a first-order linear model using some stepwise selection criteria, and choosing those that are chosen within the model.

Usage

```
AVS(x, fx, scale = 0, k = 2, trace = 0)
```

Arguments

`x`	the set of training points in the input space, given as a dataframe.
`fx`	experiment/model evaluation at the training points, given as a vector.
`scale`	specifies the estimate of the error variance. The default, `0`, indicates that this value is to be selected via maximum likelihood. `scale_maxlrm1` indicates that it is to be estimated as the variance parameter of the full model.
`k`	allows specification of the k parameter of the information criterion. The default, `2`, is AIC. `log(n)` is BIC. Can also specify strings `"BIC"` or `"logn/2"`.
`trace`	if positive, information is printed during the running of `step`. Larger values may give more detailed information.

Value

`stepwise_selected_model`
the stepwise selected model from `step`

`active_variables`
the active variables, given by those selected for `stepwise_selected_model`

See Also

`step`

Examples

```
data(USArrests)
X <- USArrests[,c("Assault", "UrbanPop")]
y <- USArrests[,"Murder"]
AVS( X, y )
```

`BLESampling` *Bayes Linear Emulator with Sampling.*

Description

Bayes Linear Emulator with Sampling.

Usage

```
BLESampling(
  object,
  EX,
  VarX,
  n_samples = 100,
  sampling_function = "normal",
  n_sd = sqrt(3),
  batch_size = 100
)
```

Arguments

`object`	an object of the type `emulate`.
`EX`	vector or matrix of expected values.
`VarX`	vector or matrix of variances, note that this should be the same dimension as `EX` as each input is sampled independently at present (this is largely an efficiency consideration).
`n_samples`	number of samples that should be sampled for each expectation and variance provided.
`sampling_function`	choice of sampling function, defaulting to normal.
`n_sd`	number of standard deviations to be used when sampling according to the uniform distribution. Using codesqrt(3), as if default, makes the uniform distribution have the mean and variance specified.
`batch_size`	the size of the batches at which to perform the predictions, default is 100.

Details

This function performs Bayes linear emulation with sampling. An uncertain input provided by each element in the vector or matrices `EX` and `VarX` is used to generate a sample from a chosen distribution with corresponding second-order specification. Each sample is run through the `predict.emulate` function, before an expected value and variance is then calculated over all of the outputs from a specific sample.

Value

`Ehx_hat`	BL adjusted expectation for h(x).
`Varhx_hat`	BL adjusted variance for h(x).

Examples

```
f <- function( x ){ c( x[2] * sin( x[1] ) + x[1] * cos( x[2] ), 2 * x[1] + 3 * x[2] / x[1] ) }
x <- matrix( runif( 20,0.2,1.2 ), ncol = 2 )
fx <- t( apply(x, 1, f) )
theta <- c( 0.4, 0.6 )
emulator <- emulate( x = x, fx = fx, CF = GaussianCF,
                     CF_para = list( theta = theta, delta = 0.0001 )  )
E_Y <- matrix( runif( 16, 0.2, 1.2), ncol = 2 )
Var_Y <- matrix( rep( 0.01, 16 ), ncol = 2 )
BLESampling( emulator, E_Y, Var_Y )
BLESampling( emulator, E_Y, Var_Y, sampling_function = "uniform" )
```

CL_CV *Cross-Validation for finding Correlation length parameters*

Description

Cross-Validation for finding Correlation length parameters

Usage

```
CL_CV(
  CF = GaussianCF,
  CF_para = list(),
  para_to_optim,
  x,
  fx,
  V_set_size = 10,
  V_splits = 10,
  G = NA,
  mean_function_model = NA,
  initial,
  lower = -Inf,
  upper = Inf,
  method = "Nelder-Mead"
)
```

Arguments

CF	A function that can be used to calculate the correlation between two matrices or dataframes of points.
CF_para	additional parameters to the function CF, along with their specified values, given as a list.
para_to_optim	a vector of the arguments to the function CF that which to assessed via LOOCV.
x	a matrix of points (given by the rows)
fx	output for each x.
V_set_size	number of points in the validation set for the cross-validation splits.
V_splits	number of cross-validation splits used.
G	model matrix. This can either be specified here or the code can calculate this given mean_function_model.

mean_function_model
: this is a linear model that is used to calculate a model matrix G.

initial
: initial settings for the parameter values to be optimised. These must either be given as a vector or a list of vectors, depending on how many arguments want to be optimised.

lower
: lower limits for the optimisation, given as a vector or list of vectors.

upper
: upper limits for the optimisation, given as a vector or list of vectors.

method
: method choice for optimisation.

Value

a list containing the found optimium correlation length parameter values.

See Also

optim

Examples

```
f <- function(x){ c( x[2] * sin( x[1] ) + x[1] * cos( x[2] ), 2 * x[1] + 3 * x[2] / x[1] ) }
x <- matrix( runif( 60, 0.2, 1.2 ), ncol = 2 )
fx <- t( apply( x, 1, f ) )
CL_CV( CF = NetworkPPBLE::GaussianCF,
       CF_para = list( delta = 0.0001 ),
       para_to_optim = "theta",
       x = x,
       fx = fx,
       mean_function_model = 1,
       initial = rep( 1, 2 )
          )
```

CL_LOOCV   *LOOCV for finding Correlation length parameters*

Description

LOOCV for finding Correlation length parameters

Usage

```
CL_LOOCV(
  CF = GaussianCF,
  CF_para = list(),
  para_to_optim,
  x,
  fx,
  G = NA,
  mean_function_model = NA,
  initial,
  lower = -Inf,
  upper = Inf,
  method = "Nelder-Mead"
)
```

Arguments

CF	A function that can be used to calculate the correlation between two matrices or dataframes of points.
CF_para	additional parameters to the function CF, along with their specified values, given as a list.
para_to_optim	a vector of the arguments to the function CF that which to assessed via LOOCV.
x	a matrix of points (given by the rows)
fx	output for each x.
G	model matrix. This can either be specified here or the code can calculate this given mean_function_model.
mean_function_model	this is a linear model that is used to calculate a model matrix G.
initial	initial settings for the parameter values to be optimised. These must either be given as a vector or a list of vectors, depending on how many arguments want to be optimised.
lower	lower limits for the optimisation, given as a vector or list of vectors.
upper	upper limits for the optimisation, given as a vector or list of vectors.
method	method choice for optimisation.

Value

a list containing the found optimium correlation length parameter values.

See Also

optim

Examples

```
f <- function(x){ c( x[2] * sin( x[1] ) + x[1] * cos( x[2] ), 2 * x[1] + 3 * x[2] / x[1] ) }
x <- matrix( runif( 20, 0.2, 1.2 ), ncol = 2 )
fx <- t( apply( x, 1, f ) )
CL_LOOCV( CF = NetworkPPBLE::GaussianCF,
          CF_para = list( delta = 0.0001 ),
          para_to_optim = "theta",
          x = x,
          fx = fx,
          mean_function_model = 1,
          initial = rep( 1, 2 )
          )
```

CL_ML *Maximum likelihood for finding Correlation length parameters*

Description

Maximum likelihood for finding Correlation length parameters

Usage

```
CL_ML(
  CF = GaussianCF,
  CF_para = list(),
  para_to_optim,
  x,
  fx,
  G = NA,
  mean_function_model = NA,
  initial,
  lower = -Inf,
  upper = Inf,
  method = "Nelder-Mead"
)
```

Arguments

CF	A function that can be used to calculate the correlation between two matrices or dataframes of points.
CF_para	additional parameters to the function CF, along with their specified values, given as a list.
para_to_optim	a vector of the names of the arguments to the function CF that which to assessed via maximum likelihood.
x	a matrix of points (given by the rows)
fx	output for each x.
G	model matrix. This can either be specifed here or the code can calculate this given mean_function_model.
mean_function_model	this is a linear model that is used to calculate a model matrix G.
initial	initial settings for the parameter values to be optimised. These must either be given as a vector or a list of vectors, depending on how many arguments want to be optimised.
lower	lower limits for the optimisation, given as a vector or list of vectors.
upper	upper limits for the optimisation, given as a vector or list of vectors.
method	method choice for optimisation.

Value

a list containing the found optimium correlation length parameter values.

See Also

optim

Examples

```
f <- function(x){ c( x[2] * sin( x[1] ) + x[1] * cos( x[2] ), 2 * x[1] + 3 * x[2] / x[1] ) }
x <- matrix( runif( 20, 0.2, 1.2 ), ncol = 2 )
fx <- t( apply( x, 1, f ) )
CL_ML( CF = NetworkPPBLE::GaussianCF,
```

```
        CF_para = list( delta = 0.0001 ),
        para_to_optim = "theta",
        x = x,
        fx = fx,
        mean_function_model = 1,
        initial = rep( 1, 2 )
         )
```

emulate *PPBLE emulate*

Description

generate an object of class emulate given a set of training points and the corresponding output of a model at those training points.

Usage

```
emulate(
  x,
  fx,
  mean_function_model = 1,
  s2 = NA,
  CF = NetworkPPBLE::GaussianCF,
  CF_para = list(),
  CF_para_optim = NULL,
  CF_para_optim_para = list()
)
```

Arguments

x — set of training points in the input space, given as the rows of a matrix or dataframe

fx — model output corresponding to the training points in x, given as a vector or matrix.

mean_function_model — model used to define the basis functions of the regression polynomial. If given as 1 (default) the regression matrix G is taken to be (1,x). If given as 2 or 3, the regression matrix G is taken to include the full quadratic or cubic terms (respectively) of the quantities in x. Otherwise a model can be specified from which the regression matrix G can be calculated.

s2 — a vector of length equal to the number of columns of fx giving the scalar variance parameter for each output component.

CF — a correlation function to be used by the emulator - it is assumed that the first two arguments of this function are to be objects (vectors, matrices or dataframes) for which the correlation function is to be computed.

CF_para — specification of the value of any additional parameters that are required for the correlation function, given in the form of a list. If a value is specified for a particular parameter, this value is used; if NA is specified, the function CF_para_optim will be used to optimise the values of those parameters; any additional parameters to CF not listed will take their default values as given by CF itself. Note that any parameters that are not given default values by CF must be listed.

`CF_para_optim` a function that optimises some of the parameters of the function `CF`.

`CF_para_optim_para`
a list of any parameters to the function `CF_para_optim` that require specifying, along with their desired specified values. Any parameters of `CF_para_optim` not listed will take their default value (as given by `CF_para_optim`). Note that any parameters that are not given default values by `CF_para_optim` must be listed.

Value

an object of class `emulate` which is a PPBLE that can be used to predict model output at further inputs, along with providing an uncertainty estimate.

Examples

```
f <- function(x){ c( x[2] * sin( x[1] ) + x[1] * cos( x[2] ), 2 * x[1] + 3 * x[2] / x[1] ) }
x <- matrix( runif( 20, 0.2, 1.2 ), ncol = 2 )
fx <- t( apply( x, 1, f ) )
theta <- c(0.4, 0.6)
emulator <- emulate( x = x, fx = fx, CF = GaussianCF,
                  CF_para = list( theta = theta, delta = 0.0001 ) )
emulator2 <- emulate( x = x, fx = fx[,1], CF_para = list( theta = theta, delta = 0.0001 ) )
X <- data.frame( x )
dimnames( X )[[2]] <- c( "X1", "X2" )
emulator3 <- emulate( x = X, fx = fx, CF = GaussianCF,
                   CF_para = list( theta = theta, delta = 0.0001 ) )
emulator4 <- emulate( x = x, fx = fx, CF = GaussianCF,
                      CF_para = list( theta = NA, delta = 0.0001 ),
                      CF_para_optim = CL_ML,
                      CF_para_optim_para = list( mean_function_model = 1,
                                                 initial = rep( 1, 2 ) ) )
```

ESplot *ESplot*

Description

ESplot

Usage

```
ESplot(
  fx,
  Efx,
  Varfx,
  n = 3,
  col_Var = "blue",
  col_E = "red",
  col_line = "black",
  main = NULL,
  sub = NULL,
  xlab = "x",
```

```
    ylab = "",
    cex = 1,
    cex.axis = 1,
    cex.lab = 1,
    cex.main = 1,
    cex.sub = 1,
    xlim = NULL,
    ylim = NULL,
    pch = 16,
    code = 3,
    angle = 90,
    length = 0.05,
    lwd = 1,
    export_type = NULL,
    filename = NA,
    height = 1000,
    width = 1000,
    mar = (c(5, 4, 4, 2) + 0.1),
    mgp = c(3, 1, 0)
  )
```

Arguments

`fx`	simulator evaluations / data, given as a vector
`Efx`	emulator expectations, given as a vector
`Varfx`	emulator variances, given as a vector
`n`	number of standard deviations either side of the mean for the range of the vertical arrow bars.
`col_Var`	colour of the error bars
`col_E`	colour of the points used to represent emulator expectation
`col_line`	colour of the line `fx` = `Efx`
`main`	main title of the plot
`sub`	subtitle of the plot
`xlab`	x-axis label
`ylab`	y-axis label
`cex`	size of the points, as given by `par`.
`cex.axis`	the magnification to be used for axis annotation relative to the current setting of `cex`.
`cex.lab`	the magnification to be used for the x and y labels relative to the current setting of `cex`.
`cex.main`	The magnification to be used for the main titles relative to the current setting of `cex`.
`cex.sub`	The magnification to be used for sub-titles relative to the current setting of `cex/`
`xlim`	x-axis limits
`ylim`	y-axis limits
`pch`	integer specifying a symbol or a single character to be used as the default in plotting points.

code	integer code, determining *kind* of arrows to be drawn.
angle	angle from the shaft of the arrow to the edge of the arrow head.
length	length of the edges of the arrow head (in inches).
lwd	graphical parameters, possible vectors. NA values in col cause the arrow to be omitted.
export_type	type of file that the image should be exported as (if required). By default this is NULL, in which case the required plot is plotted in the R interface.
filename	name of the exported file.
height	height of the image
width	width of the image
mar	a numerical vector of the form c(bottom,left,top,right) which gives the number of lines of margin to be specified on the four sides of the plot. The default is c(5,4,4,2) + 0.1.
mgp	The margin line (in mex units) for the axis title, axis labels and axis line. Note that mgp[1] affects title whereas mgp[2:3] affect axis. The default is c(3, 1, 0).

Value

A plot, either exported or in the R interface itself.

Examples

```
fx <- stats::runif( 10 )
Efx <- stats::runif( 10 )
Varfx <- abs( stats::rnorm( 10, sd = 0.05 ) )
ESplot( fx = fx, Efx = Efx, Varfx = Varfx )
```

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 )
```

GaussianCFUI *Gaussian Correlation Function with Uncertain Inputs*

Description

Gaussian Correlation Function with Uncertain Inputs

Usage

```
GaussianCFUI(EX, EY = EX, VarX = 0, VarY = 0, CovXY = 0, theta, delta = 0)
```

Arguments

`EX`	a vector, matrix or dataframe of the expected values of X
`EY`	a vector, matrix or dataframe of the expected values of Y
`VarX`	a vector, matrix or dataframe of the variances of X
`VarY`	a vector, matrix or dataframe of the variances of Y
`CovXY`	a matrix or array of the covariances between the points X and Y.
`theta`	a vector of correlation length parameter values (one for each column of EX).
`delta`	an (optional) scalar nugget parameter.

Value

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

Examples

```
EX <- matrix( rnorm( 10 ), ncol = 2 )
EY <- matrix( runif( 6 ), ncol = 2 )
VarX <- matrix ( rep( 0.01, 10 ), ncol = 2 )
VarY <- matrix ( rep( 0.01, 6 ), ncol = 2 )
theta <- c( 0.5, 0.8 )
GaussianCFUI( EX, EY,  VarX, VarY, theta = theta )
GaussianCFUI( as.data.frame( EX ), EY, VarX, VarY, theta = theta )
```

GLSbeta *Generalised Least Squares Estimate for Beta*

Description

Generalised Least Squares Estimate for Beta

Usage

```
GLSbeta(fx, C, G)
```

Arguments

`fx`	a set of n model runs, each with a k-component model output, given as an n by k matrix or n-vector.
`C`	correlation matrix between the model runs `fx`.
`G`	model matrix for the set of runs `fx`.

Value

`betahatGLS`	Generalised Least Squares estimate for beta coefficients
`Q`	Q matrix - corresponding to G^TC^-1G

Examples

```
f <- function( x ){ x[2] * sin( x[1] ) + x[1] * cos( x[2] ) }
x <- matrix( runif( 20, 0.2, 1.2 ), ncol = 2 )
fx <- apply( x, 1, f )
C <- GaussianCF( x, theta = c(0.4, 0.6) )
G <- cbind( rep( 1, 10 ), x )
GLSbeta( fx, C, G )
```

GPTimeSeriesSampling *Generate a psuedo-GP sample from a BLE time series.*

Description

Generate a psuedo-GP sample from a BLE time series.

Usage

```
GPTimeSeriesSampling(
  Efx,
  Varfx,
  times,
  theta,
  subtimes = times,
  nz = 10,
  non_negative = FALSE
)
```

Arguments

Efx	Expected value of f(x) across time.
Varfx	Variance of f(x) across time.
times	Values of time (t) for which BLE was evaluated.
theta	Correlation length parameter across time.
subtimes	Vector of subtimes for the purposes of sampling. Useful if the length of times makes MVN sampling too computationally intensive.
nz	Number of samples required for each x-value.
non_negative	if true, zeroes any negative samples.

Value

A matrix of sampled time series, with nz samples for each row of Efx.

Examples

```
beta_sample <- runif( 5 )
times <- seq( from = 0 , to = 1, by = 0.1 )
Efx <- kronecker( beta_sample, t( times ) )
Varfx <- kronecker( rep( 1, 5 ), t( ( 0.5 - times )^2 ) )
theta <- 1
GPsamp <- GPTimeSeriesSampling( Efx = Efx, Varfx = Varfx, times = times, theta = theta )
par( mfrow = c(1, 1) )
graphics::matplot( times, t( GPsamp ), type = "l", lty = 1 )
```

MaternCF *Matern Correlation Function*

Description

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

Usage

```
MaternCF(X, Y = X, kappa, nu)
```

Arguments

X	a vector, matrix or dataframe
Y	a vector, matrix or dataframe
kappa	range paraneter
nu	shape parameter

Value

Matern 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 )
kappa <- 10
nu <- 1/5
MaternCF( X, Y, kappa, nu )
MaternCF( as.data.frame(X), Y, kappa, nu )
```

`mean_sd_plot` *Mean and standard deviation plot (against continuous x-variable).*

Description

Plot the mean and standard deviation curves over a continuous x-variable (typically time).

Usage

```
mean_sd_plot(
  x,
  fx = NULL,
  Efx,
  Varfx,
  col_Efx = 1,
  col_fx = 2,
  col_sd = 3,
  lwd = 2,
  lty_Efx = 1,
  lty_fx = 1,
  lty_sd = 1,
  ylim = c(0, max(Efx)),
  xlim = c(min(x), max(x)),
  xlab = "",
  ylab = "",
  ca = 1,
  cl = 1,
  main = rep("", nrow(Efx)),
  export_type = NULL,
  filename = NA,
  height = 1000,
  width = 1000,
  mar = (c(5, 4, 4, 2) + 0.1),
  mgp = c(3, 1, 0),
  mfrow = c(1, 1)
)
```

Arguments

`x`	Vector of x-values (typically representing say, time).
`fx`	Matrix of true model output across `x`, if known (by default `NULL`.
`Efx`	Matrix of emulator expectation across `x`.

Varfx	Matrix of emulator variance across x.
col_Efx	Colour of mean line.
col_fx	Colour of true model line.
col_sd	Colour of standard deviation line.
lwd	line widths.
lty_Efx	Line type for mean line.
lty_fx	Line type of true model line.
lty_sd	Line type of standard deviation line.
ylim	Limits on y-axis - can be given as an n by 2 matrix if different limits are required for each plot.
xlim	Limits on x-axis - can be given as an n by 2 matrix if different limits are required for each plot.
xlab	x-axis label.
ylab	y-axis label.
ca	Font size of axis.
cl	Font size of labels.
main	vector of plot titles if required.
export_type	type of file that the image should be exported as (if required). By default this is NULL, in which case the required plot is plotted in the R interface.
filename	name of the exported file.
height	height of the image
width	width of the image
mar	a numerical vector of the form c(bottom,left,top,right) which gives the number of lines of margin to be specified on the four sides of the plot. The default is c(5,4,4,2) + 0.1.
mgp	The margin line (in mex units) for the axis title, axis labels and axis line. Note that mgp[1] affects title whereas mgp[2:3] affect axis. The default is c(3, 1, 0).
mfrow	The number of rows and columns to divide the device window into, given as a vector of two elements.

Value

Plot.

Examples

```
x = 1:100
Efx = matrix( c( x^(2.1/2), x^(2.2/2) ), nrow = 2, byrow=TRUE )
Varfx = matrix( c(x,x), nrow = 2, byrow = TRUE)
mean_sd_plot( x = x, Efx = Efx, Varfx = Varfx )
```

model_matrix — *Construct model matrix for any dataframe and model.*

Description

Construct model matrix for any dataframe and model.

Usage

```
model_matrix(x, model = 1)
```

Arguments

`x`	set of training points in the input space, given as the rows of a matrix or dataframe.
`model`	model used to define the basis functions of the regression polynomial. If given as `1` (default) the regression matrix G is taken to be (1,x). If given as `2` or `3`, the model matrix `MM` is taken to include the full quadratic or cubic terms (respectively) of the quantities in x. Otherwise a model can be specified from which the regression matrix G can be calculated.

Value

Model matrix.

Examples

```
x <- matrix( runif( 20, 0.2, 1.2 ), ncol = 2 )
model_matrix( x )
model_matrix( x, 2 )
```

partition.test — *Test Data Partitions*

Description

Find the partitions of a set of test data as given by a Treed Gaussian process having used, for example, btgp

Usage

```
partition.test(x, tgp_object)
```

Arguments

`x`	Matrix of input points to partition.
`tgp_object`	An object of the class `tgp`.

Value

Submatrices of `x`, partitioned according to the partitions of `tgp_object`.

See Also

btgp

Examples

```
# construct some 1-d nonstationary data - this exammple uses one of
# those found in the tgp package itself.
X <- seq( 0, 20, length = 100 )
XX <- seq( 0, 20, length = 99 )
Z <- ( sin( pi * X / 5 ) + 0.2 * cos( 4 * pi * X / 5 ) ) * ( X <= 9.6 )
lin <- X > 9.6;
Z[lin] <- -1 + X[lin] / 10
Z <- Z + rnorm( length( Z ), sd = 0.1 )
out <- tgp::btgp( X = X, Z = Z, XX = XX )  # use a treed GP
plot( out )  # plot the surface
tgp::tgp.trees( out )
partition.test( x = matrix( XX, ncol = 1 ), tgp_object = out )
```

PeriodicGaussianCF *Periodic Gaussian Correlation Function*

Description

Calculate the Gaussian correlation function between the points in (given by the rows of) two matrices, given that there is periodicity in the values of (some of) the inputs.

Usage

```
PeriodicGaussianCF(X, Y = X, theta, p = rep(NA, NCOL(X)), 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).
p	vector of period values for each of the inputs. If there is no period then the corresponding element of this vector should be given as NA.
delta	an (optional) scalar nugget parameter.

Value

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

See Also

GaussianCF

Examples

```
X <- matrix( rnorm( 10 ), ncol = 2 )
Y <- matrix( runif( 6 ), ncol = 2 )
theta <- c( 0.5, 0.8 )
GaussianCF( X, Y, theta = theta )
# if p is not specified, then the results should be the same as above.
PeriodicGaussianCF( X, Y, theta = theta )
# compare if we have a period of 1 for each input.
PeriodicGaussianCF( X, Y, theta = theta, p = rep( 1, 2 ) )
# or a period for just one of the inputs.
PeriodicGaussianCF( X, Y, theta = theta, p = c( NA, 0.5 ) )
```

PMarrows *Plot plus-and-minus standard deviation Arrows*

Description

Plot plus-and-minus standard deviation Arrows

Usage

```
PMarrows(
  x,
  mean,
  Var,
  n = 3,
  code = 3,
  angle = 90,
  length = 0.05,
  col = "black",
  lwd = 1
)
```

Arguments

`x`	vector of x coordinates
`mean`	vector of expected values
`Var`	vector of variances
`n`	number of standard deviations for the vertical arrows to be covering.
`code`	integer code, determining *kind* of arrows to be drawn.
`angle`	angle from the shaft of the arrow to the edge of the arrow head.
`length`	length of the edges of the arrow head (in inches).
`col`	graphical parameters, possible vectors. `NA` values in `col` cause the arrow to be omitted.
`lwd`	graphical parameters, possible vectors. `NA` values in `col` cause the arrow to be omitted.

Value

plots some arrows

Examples

```
x <- stats::runif( 10 )
mean <- stats::runif( 10 )
Var <- abs( stats::rnorm( 10, sd = 0.05 ) )
plot( x = x, y = mean, type="n" )
PMarrows( x = x, mean = mean, Var = Var )
```

PowerCF *Power Correlation Function*

Description

Power Correlation Function

Usage

```
PowerCF(X, Y = X, theta, P = 2, delta = 0)
```

Arguments

X	vector, matrix or dataframe
Y	a vector, matrix or dataframe
theta	a vector of correlation length parameter values (one for each column of X).
P	a scalar power parameter, note the default value of 2 yields the Gaussian Correlation Function.
delta	an (optional) scalar nugget parameter.

Value

Power 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 )
P <- 1.9
PowerCF( X, Y, theta, P, delta = 0.001 )
```

predict.emulate — *PPBLE predict (emulate)*

Description

predict the value of a model at some new input locations, along with a measure of uncertainty, using a PPBLE of class `emulate`.

Usage

```
## S3 method for class 'emulate'
predict(object, x = "training", batch_size = 100, ...)
```

Arguments

`object`	an object of class `emulate`
`x`	a set of training runs at which to make predictions using the `emulate` object, given as a vector, matrix or dataframe (the same type as xv)
`batch_size`	the size of the batches at which to perform the predictions, default is 100.
`...`	additional arguments (although not quite sure what these may be)

Value

`Efx`	BL adjusted expectation for f(x)
`Varfx`	BL adjusted variance for f(x)

Examples

```
f <- function(x){ x[2] * sin( x[1] ) + x[1] * cos( x[2] ) }
x <- matrix( runif( 20, 0.2, 1.2 ), ncol = 2 )
fx <- apply( x, 1, f )
theta <- c( 0.4, 0.6 )
emulator <- emulate( x = x, fx = fx, CF = GaussianCF,
            CF_para = list( theta = theta, delta = 0.0001 )  )
y <- matrix( runif( 16, 0.2, 1.2 ), ncol = 2 )
predict( emulator, y )
```

UI_predict — *PPBLE predict with Uncertain Inputs*

Description

predict the value of a model at some new input locations, along with a measure of uncertainty, using a PPBLE of class `emulate`. We assume that the points at which we wish to predict are uncertain, given by an expected value and a variance (Bayes linear). Having said this, we assume that the emulaor training points were fixed and known, hence we can use a regular object of class `emulate`.

Usage

```
UI_predict(object, EX, VarX, UICF, batch_size = 100)
```

Arguments

object	an object of class emulate
EX	vector or matrix of expected values.
VarX	vector or matrix of variances.
UICF	an Uncertain Inputs correlation function to be used. It is assumed that this is an extension of CF used by emulate when the emulator was built. As such, it assumes that any additional parameters are the same. It also assumes that the first four objects of this correlation function are the expectations followed by the variances of the two matrices of points at which to run the correlation function.
batch_size	the size of the batches at which to perform the predictions, default is 100.

Value

EfX	BL adjusted expectation for f(X)
VarfX	BL adjusted variance for f(X)

Examples

```
f <- function( x ){ c( x[2] * sin( x[1] ) + x[1] * cos( x[2] ), 2 * x[1] + 3 * x[2] / x[1] ) }
x <- matrix( runif( 20, 0.2, 1.2 ), ncol = 2 )
fx <- t( apply( x, 1, f ) )
theta <- c( 0.4, 0.6 )
emulator <- emulate( x = x, fx = fx, CF = GaussianCF,
                  CF_para = list( theta = theta, delta = 0.0001 ) )
E_Y <- matrix( runif( 16, 0.2, 1.2), ncol = 2 )
Var_Y <- matrix( rep( 0.01, 16 ), ncol = 2 )
UI_predict( emulator, E_Y, Var_Y, UICF = GaussianCFUI )
```

Index