Package ‘bliss’

April 18, 2024

Title Bayesian Functional Linear Regression with Sparse Step Functions

Version 1.0.5

Author Paul-Marie Grollemund [aut, cre],
Isabelle Sanchez [ctr],
Meili Baragatti [ctr]

Maintainer Paul-Marie Grollemund <paul.marie.grollemund@gmail.com>

Description A method for the Bayesian functional linear regression model (scalar-on-function), including two estimators of the coefficient function and an estimator of its support.
A representation of the posterior distribution is also available. Grollemund P-M., Abraham C., Baragatti M., Pudlo P. (2019) <doi:10.1214/18-BA1095>.

License GPL-3

Depends R (>= 3.3.0)

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

LazyData TRUE

URL https://github.com/pmgrollemund/bliss

BugReports https://github.com/pmgrollemund/bliss/issues

Imports Rcpp, MASS

Suggests rmarkdown, knitr, RColorBrewer

RoxygenNote 7.1.2

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-04-18 07:50:03 UTC

R topics documented:

`BIC_model_choice` *BIC_model_choice*

Description

Model selection with BIC criterion.

Usage

```
BIC_model_choice(Ks, iter, data, verbose = T)
```

Arguments

`Ks`	a numerical vector containing the K values.
`iter`	an integer, the number of iteration for each run of `fit_Bliss`.
`data`	a list containing: **Q** an integer, the number of functional covariates. **y** a numerical vector, the outcomes. **x** a list of matrices, the qth matrix contains the observations of the qth functional covariate at time points given by `grids`. **grids** a list of numerical vectors, the qth vector is the grid of time points for the qth functional covariate.
`verbose`	write stuff if TRUE (optional).

Value

A numerical vector, the BIC values for the Bliss model for different K value.

Examples

```
param_sim <- list(Q=1,n=100,p=c(50),grids_lim=list(c(0,1)))
data      <- sim(param_sim,verbose=TRUE)
iter = 1e2
Ks <- 1:5

res_BIC <- BIC_model_choice(Ks,iter,data)
plot(res_BIC,xlab="K",ylab="BIC")
```

`bliss` *bliss: Bayesian functional Linear regression with Sparse Step functions*

Description

A method for the Bayesian Functional Linear Regression model (functions-on-scalar), including two estimators of the coefficient function and an estimator of its support. A representation of the posterior distribution is also available.

`Bliss_Gibbs_Sampler` *Bliss_Gibbs_Sampler*

Description

A Gibbs Sampler algorithm to sample the posterior distribution of the Bliss model.

Usage

```
Bliss_Gibbs_Sampler(data, param, verbose = FALSE)
```

Arguments

`data`	a list containing: **Q** an integer, the number of functional covariates. **y** a numerical vector, the outcome values `y_i`. **x** a list of matrices, the qth matrix contains the observations of the qth functional covariate at time points given by `grids`. **grids** a list of numerical vectors, the qth vector is the grid of time points for the qth functional covariate.
`param`	a list containing: **iter** an integer, the number of iterations of the Gibbs sampler algorithm. **K** a vector of integers, corresponding to the numbers of intervals for each covariate. **p** an integer, the number of time points. **basis** a character vector (optional). The possible values are "uniform" (default), "epanechnikov", "gauss" and "triangular" which correspond to different basis functions to expand the coefficient function and the functional covariates **phi_l** a numerical (optional). An hyperparameters related to the exponential prior on the length of the intervals. Lower values promotes wider intervals.
`verbose`	write stuff if TRUE (optional).

Value

a list containing :

trace a matrix, the trace of the Gibbs Sampler.

param a list containing parameters used to run the function.

Examples

```
# May take a while
param_sim <- list(Q=1,n=25,p=50,grids_lim=list(c(0,1)),iter=1e4,K=2)
data_sim <- sim(param_sim,verbose=FALSE)
```

```
res_Bliss_Gibbs_Sampler <- Bliss_Gibbs_Sampler(data_sim,param_sim)
theta_1 <- res_Bliss_Gibbs_Sampler$trace[1,]
theta_1
# Resultat for few iterations
param_sim <- list(Q=1,n=25,p=50,grids_lim=list(c(0,1)),iter=5e2,K=2)
data_sim <- sim(param_sim,verbose=FALSE)
res_Bliss_Gibbs_Sampler <- Bliss_Gibbs_Sampler(data_sim,param_sim)
theta_1 <- res_Bliss_Gibbs_Sampler$trace[1,]
theta_1
```

`Bliss_Simulated_Annealing`

Bliss_Simulated_Annealing

Description

A Simulated Annealing algorithm to compute the Bliss estimate.

Usage

```
Bliss_Simulated_Annealing(
  beta_sample,
  normalization_values,
  param,
  verbose = FALSE
)
```

Arguments

`beta_sample`	a matrix. Each row is a coefficient function computed from the posterior sample.
`normalization_values`	a matrix given by the function `Bliss_Gibbs_Sampler`.
`param`	a list containing: **grid** a numerical vector, the time points. **K** an integer, the number of intervals. **basis** a character vector (optional). The possible values are "uniform" (default), "epanechnikov", "gauss" and "triangular" which correspond to different basis functions to expand the coefficient function and the functional covariates **burnin** an integer (optional), the number of iteration to drop from the posterior sample. **iter_sann** an integer (optional), the number of iteration of the Simulated Annealing algorithm. **k_max** an integer (optional), the maximal number of intervals for the Simulated Annealing algorithm.

- **l_max** an integer (optional), the maximal interval length for the Simulated Annealing algorithm.
- **Temp_init** a nonnegative value (optional), the initial temperature for the cooling function of the Simulated Annealing algorithm.

`verbose` write stuff if TRUE (optional).

Value

a list containing:

- **Bliss_estimate** a numerical vector, corresponding to the Bliss estimate of the coefficient function.
- **Smooth_estimate** a numerical vector, which is the posterior expectation of the coefficient function for each time points.
- **trace** a matrix, the trace of the algorithm.
- **argmin** an integer, the index of the iteration minimizing the Bliss loss.
- **difference** a numerical vector, the difference between the Bliss estimate and the smooth estimate.
- **sdifference** a numerical vector, a smooth version of `difference`.

Examples

```
data(data1)
data(param1)
param1$grids<-data1$grids
# result of res_bliss1<-fit_Bliss(data=data1,param=param1)
data(res_bliss1)
beta_sample <- compute_beta_sample(posterior_sample=res_bliss1$posterior_sample,
                                   param=param1,Q=1)
param_test<-list(grid=param1$grids[[1]],iter=1e3,K=2)
test<-Bliss_Simulated_Annealing(beta_sample[[1]],
                 res_bliss1$posterior_sample$param$normalization_values[[1]],
                 param=param_test)
ylim <- range(range(test$Bliss_estimate),range(test$Smooth_estimate))
plot(param_test$grid,test$Bliss_estimate,type="l",ylim=ylim)
lines(param_test$grid,test$Smooth_estimate,lty=2)
```

`build_Fourier_basis` *build_Fourier_basis*

Description

Define a Fourier basis to simulate functional covariate observations.

Usage

```
build_Fourier_basis(grid, dim, per = 2 * pi)
```

Arguments

`grid`	a numerical vector.
`dim`	a numerical value. It corresponds to `dim(basis)/2`.
`per`	a numerical value which corresponds to the period of the sine and cosine functions.

Details

See the `sim_x` function.

Value

a matrix. Each row is an functional observation evaluated on the `grid` time points.

Examples

```
# See the function \code{sim_x}.
```

`change_grid` *change_grid*

Description

Compute a function (evaluated on a grid) on a given (finer) grid.

Usage

```
change_grid(fct, grid, new_grid)
```

Arguments

`fct`	a numerical vector, the function to evaluate on the new grid.
`grid`	a numerical vector, the initial grid.
`new_grid`	a numerical vector, the new grid.

Value

a numerical vector, the approximation of the function on the new grid.

Examples

```
grid <- seq(0,1,l=1e1)
new_grid <- seq(0,1,l=1e2)
fct <- 3*grid^2 + sin(grid*2*pi)
plot(grid,fct,type="o",lwd=2,cex=1.5)
lines(new_grid,change_grid(fct,grid,new_grid),type="o",col="red",cex=0.8)
```

choose_beta *choose_beta*

Description

Compute a coefficient function for the Function Linear Regression model.

Usage

```
choose_beta(param)
```

Arguments

`param`	a list containing: **grid** a numerical vector, the time points. **p** a numerical value, the length of the vector `grid`. **shape** a character vector: "smooth", "random_smooth", "simple", "simple_bis", "random_simple", "sinusoid", "flat_sinusoid" and "sharp"

Details

Several shapes are available.

Value

A numerical vector which corresponds to the coefficient function at given times points (`grid`).

Examples

```
### smooth
param <- list(p=100,grid=seq(0,1,length=100),shape="smooth")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="l")
### random_smooth
param <- list(p=100,grid=seq(0,1,length=100),shape="random_smooth")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="l")
### simple
param <- list(p=100,grid=seq(0,1,length=100),shape="simple")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="s")
### simple_bis
param <- list(p=100,grid=seq(0,1,length=100),shape="simple_bis")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="s")
### random_simple
param <- list(p=100,grid=seq(0,1,length=100),shape="random_simple")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="s")
```

```
### sinusoid
param <- list(p=100,grid=seq(0,1,length=100),shape="sinusoid")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="l")
### flat_sinusoid
param <- list(p=100,grid=seq(0,1,length=100),shape="flat_sinusoid")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="l")
### sharp
param <- list(p=100,grid=seq(0,1,length=100),shape="sharp")
beta_function <- choose_beta(param)
plot(param$grid,beta_function,type="l")
```

---

compute_beta_posterior_density

compute_beta_posterior_density

Description

Compute the posterior density of the coefficient function.

Usage

```
compute_beta_posterior_density(beta_sample, param, verbose = FALSE)
```

Arguments

`beta_sample`	a matrix. Each row is a coefficient function computed from the posterior sample.
`param`	a list containing: **grid** a numerical vector, the time points. **lims_estimate** a numerical vector, the time points. **burnin** an integer (optional), the number of iteration to drop from the Gibbs sample. **lims_kde** an integer (optional), correspond to the `lims` option of the `kde2d` funtion. **new_grid** a numerical vector (optional) to compute beta sample on a different grid. **thin** an integer (optional) to thin the posterior sample.
`verbose`	write stuff if TRUE (optional).

Details

The posterior densities correponds to approximations of the marginal posterior distribitions (of beta(t) for each t). The sample is thinned in order to reduce the correlation and the computational time of the function `kde2d`.

Value

An approximation of the posterior density on a two-dimensional grid (corresponds to the result of the kde2d function).

Examples

```
library(RColorBrewer)
data(data1)
data(param1)
# result of res_bliss1<-fit_Bliss(data=data1,param=param1)
data(res_bliss1)
q <- 1
param_beta_density <- list(grid= data1[["grids"]][[q]],
                           iter= param1[["iter"]],
                           p   = param1[["p"]][q],
                           n        = length(data1[["y"]]),
                           thin     = param1[["thin"]],
                           burnin   = param1[["burnin"]],
                           lims_kde = param1[["lims_kde"]][[q]],
                           new_grid = param1[["new_grids"]][[q]],
                           lims_estimate = range(res_bliss1$Smooth_estimate[[q]]))
density_estimate <- compute_beta_posterior_density(res_bliss1$beta_sample[[q]],param_beta_density)
image(density_estimate$grid_t,
      density_estimate$grid_beta_t,
      density_estimate$density,col=rev(heat.colors(100)))
```

`compute_beta_sample` *compute_beta_sample*

Description

Compute the posterior coefficient function from the posterior sample.

Usage

```
compute_beta_sample(posterior_sample, param, Q, verbose = FALSE)
```

Arguments

`posterior_sample`	a list provided by the function `Bliss_Gibbs_Sampler`.
`param`	a list containing: **K** a vector of integers, corresponding to the numbers of intervals for each covariate. **grids** a numerical vector, the observation time points.

basis a vector of characters (optional) among : "uniform" (default), "epanechnikov", "gauss" and "triangular" which correspond to different basis functions to expand the coefficient function and the functional covariates.

`Q`	numeric
`verbose`	write stuff if TRUE (optional).

Value

return a matrix containing the coefficient function posterior sample.

Examples

```
library(RColorBrewer)
data(data1)
data(param1)
param1$grids<-data1$grids
# result of res_bliss1<-fit_Bliss(data=data1,param=param1)
data(res_bliss1)
beta_sample <- compute_beta_sample(posterior_sample=res_bliss1$posterior_sample,
                                   param=param1,Q=1)
indexes <- sample(nrow(beta_sample[[1]]),1e2,replace=FALSE)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(1e2)
matplot(param1$grids[[1]],t(beta_sample[[1]][indexes,]),type="l",lty=1,col=cols,
xlab="grid",ylab="")
```

`compute_chains_info` *compute_chains_info*

Description

Compute summaries of Gibbs Sampler chains.

Usage

```
compute_chains_info(chain, param)
```

Arguments

`chain` a list given by the `Bliss_Gibbs_Sampler` function.

`param` a list containing:

K a vector of integers, corresponding to the numbers of intervals for each covariate.

grids a numerical vector, the observation time points.

basis a vector of characters (optional) among : "uniform" (default), "epanechnikov", "gauss" and "triangular" which correspond to different basis functions to expand the coefficient function and the functional covariates.

Value

Return a list containing the estimates of mu and sigma_sq, the Smooth estimate and the chain autocorrelation for mu, sigma_sq and beta.

Examples

```
param_sim <- list(Q=1,
                  n=100,
                  p=c(50),
                  grids_lim=list(c(0,1)))
data <- sim(param_sim,verbose=TRUE)

param <- list(iter=5e2,
              K=c(3),
              n_chains = 3)
res_bliss <- fit_Bliss(data,param,verbose=TRUE,compute_density=FALSE,sann=FALSE)

param$grids <- data$grids
chains_info1 <- compute_chains_info(res_bliss$chains[[1]],param)
chains_info2 <- compute_chains_info(res_bliss$chains[[2]],param)
chains_info3 <- compute_chains_info(res_bliss$chains[[3]],param)

# Smooth estimates
ylim <- range(range(chains_info1$estimates$Smooth_estimate),
range(chains_info2$estimates$Smooth_estimate),
range(chains_info3$estimates$Smooth_estimate))
plot(data$grids[[1]],chains_info1$estimates$Smooth_estimate,type="l",ylim=ylim,
xlab="grid",ylab="")
lines(data$grids[[1]],chains_info2$estimates$Smooth_estimate,col=2)
lines(data$grids[[1]],chains_info3$estimates$Smooth_estimate,col=3)

# Autocorrelation
plot(chains_info1$autocorr_lag[,1],type="h")
```

---

compute_random_walk *compute_random_walk*

Description

Compute a (Gaussian) random walk.

Usage

```
compute_random_walk(n, p, mu, sigma, start = rep(0, n))
```

Arguments

`n`	an integer, the number of random walks.
`p`	an integer, the length of the random walks.
`mu`	a numerical vector, the mean of the random walks.
`sigma`	a numerical value which is the standard deviation of the gaussian distribution used to compute the random walks.
`start`	a numerical vector (optional) which is the initial value of the random walks.

Details

See the `sim_x` function.

Value

a matrix where each row is a random walk.

Examples

```
# see the sim_x() function.
```

`compute_starting_point_sann`

compute_starting_point_sann

Description

Compute a starting point for the Simulated Annealing algorithm.

Usage

```
compute_starting_point_sann(beta_expe)
```

Arguments

`beta_expe`	a numerical vector, the expectation of the coefficient function posterior sample.

Value

a matrix with 3 columns : "m", "l" and "b". The two first columns define the begin and the end of the intervals and the third gives the mean values of each interval.

Examples

```
data(res_bliss1)
mystart<-compute_starting_point_sann(apply(res_bliss1$beta_sample[[1]],2,mean))
```

corr_matrix *corr_matrix*

Description

Compute an autocorrelation matrix.

Usage

```
corr_matrix(diagonal, ksi)
```

Arguments

diagonal	a numerical vector corresponding to the diagonal.
ksi	a numerical value, related to the correlation.

Value

a symmetric matrix.

Examples

```
### Test 1 : weak autocorrelation
ksi     <- 1
diagVar <- abs(rnorm(100,50,5))
Sigma   <- corr_matrix(diagVar,ksi^2)
persp(Sigma)
### Test 2 : strong autocorrelation
ksi     <- 0.2
diagVar <- abs(rnorm(100,50,5))
Sigma   <- corr_matrix(diagVar,ksi^2)
persp(Sigma)
```

data1 *a list of data*

Description

A data object for bliss model

Usage

```
data1
```

Format

a list of data

Q the number of functional covariates

y y coordinate

x x coordinate

betas the coefficient function used to generate the data

grids the grid of the observation times

`determine_intervals` *determine_intervals*

Description

Determine for which intervals a function is nonnull.

Usage

```
determine_intervals(beta_fct)
```

Arguments

`beta_fct`	a numerical vector.

Value

a matrix with 3 columns : "begin", "end" and "value". The two first columns define the begin and the end of the intervals and the third gives the mean values of each interval.

Examples

```
data(data1)
data(param1)
# result of res_bliss1<-fit_Bliss(data=data1,param=param1)
data(res_bliss1)
intervals <- determine_intervals(res_bliss1$Bliss_estimate[[1]])
plot(data1$grids[[1]],res_bliss1$Bliss_estimate[[1]],type="s")
for(k in 1:nrow(intervals)){
   segments(data1$grids[[1]][intervals[k,1]],intervals[k,3],
           data1$grids[[1]][intervals[k,2]],intervals[k,3],col=2,lwd=4)
}
```

`dposterior` *dposterior*

Description

Compute (non-normalized) posterior densities for a given parameter set.

Usage

```
dposterior(posterior_sample, data, theta = NULL)
```

Arguments

`posterior_sample`	a list given by the `Bliss_Gibbs_Sampler` function.
`data`	a list containing **y** a numerical vector, the outcomes. **x** a list of matrices, the qth matrix contains the observations of the qth functional covariate at time points given by `grids`.
`theta`	a matrix or a vector which contains the parameter set.

Details

If the `theta` is NULL, the posterior density is computed from the MCMC sample given in the `posterior_sample`.

Value

Return the (log) posterior density, the (log) likelihood and the (log) prior density for the given parameter set.

Examples

```
data(data1)
data(param1)
# result of res_bliss1<-fit_Bliss(data=data1,param=param1)
data(res_bliss1)
# Compute the posterior density of the MCMC sample :
res_poste <- dposterior(res_bliss1$posterior_sample,data1)
```

fit_Bliss *fit_Bliss*

Description

Fit the Bayesian Functional Linear Regression model (with Q functional covariates).

Usage

```
fit_Bliss(
  data,
  param,
  compute_density = TRUE,
  sann = TRUE,
  support_estimate = TRUE,
  verbose = FALSE
)
```

Arguments

`data`	a list containing: **Q** an integer, the number of functional covariates. **y** a numerical vector, the outcomes. **x** a list of matrices, the qth matrix contains the observations of the qth functional covariate at time points given by `grids`. **grids** a list of numerical vectors, the qth vector is the grid of time points for the qth functional covariate.
`param`	a list containing: **iter** an integer, the number of iterations of the Gibbs sampler algorithm. **K** a vector of integers, corresponding to the numbers of intervals for each covariate. **basis** a character vector (optional). The possible values are "uniform" (default), "epanechnikov", "gauss" and "triangular" which correspond to different basis functions to expand the coefficient function and the functional covariates **burnin** an integer (optional), the number of iteration to drop from the posterior sample. **iter_sann** an integer (optional), the number of iteration of the Simulated Annealing algorithm. **k_max** an integer (optional), the maximal number of intervals for the Simulated Annealing algorithm. **l_max** an integer (optional), the maximal interval length for the Simulated Annealing algorithm. **lims_kde** an integer (optional), correspond to the `lims` option of the `kde2d` funtion.

- **n_chains** an integer (optional) which corresponds to the number of Gibbs sampler runs.
- **new_grids** a list of Q vectors (optional) to compute beta samples on different grids.
- **Temp_init** a nonnegative value (optional), the initial temperature for the cooling function of the Simulated Annealing algorithm.
- **thin** an integer (optional) to thin the posterior sample.
- **times_sann** an integer (optional), the number of times the algorithm will be executed

`compute_density`	a logical value. If TRUE, the posterior density of the coefficient function is computed. (optional)
`sann`	a logical value. If TRUE, the Bliss estimate is computed with a Simulated Annealing Algorithm. (optional)
`support_estimate`	a logical value. If TRUE, the estimate of the coefficient function support is computed. (optional)
`verbose`	write stuff if TRUE (optional).

Value

return a list containing:

- **alpha** a list of Q numerical vector. Each vector is the function alpha(t) associated to a functional covariate. For each t, alpha(t) is the posterior probabilities of the event "the support covers t".
- **beta_posterior_density** a list of Q items. Each item contains a list containing information to plot the posterior density of the coefficient function with the `image` function.
  - `grid_t` a numerical vector: the x-axis.
  - `grid_beta_t` a numerical vector: the y-axis.
  - `density` a matrix: the z values.
  - `new_beta_sample` a matrix: beta sample used to compute the posterior densities.
- **beta_sample** a list of Q matrices. The qth matrix is a posterior sample of the qth functional covariates.
- **Bliss_estimate** a list of numerical vectors corresponding to the Bliss estimates of each functional covariates.
- **chains** a list of `posterior_sample`. `chains` is `NULL` if `n_chains=1`.
- **chains_info** a list for each chain providing: a mu estimate, a sigma_sq estimate, the Smooth estimate of the coefficient function and the autocorrelation of the Markov Chain.
- **data** a list containing the data.
- **posterior_sample** a list of information about the posterior sample: the trace matrix of the Gibbs sampler, a list of Gibbs sampler parameters and the posterior densities.
- **support_estimate** a list of support estimates of each functional covariate.
- **support_estimate_fct** another version of the support estimates.
- **trace_sann** a list of Q matrices which are the trace of the Simulated Annealing algorithm.

Examples

```
# see the vignette BlissIntro.
```

---

image_Bliss *image_Bliss*

Description

Plot an approximation of the posterior density.

Usage

```
image_Bliss(beta_posterior_density, param = list(), q = 1)
```

Arguments

`beta_posterior_density`	a list. The result of the function `compute_beta_posterior_density`.
`param`	a list containing: (optional) **cols** a vector of colors for the function image. **main** an overall title for the plot. **xlab** a title for the x axis. **ylab** a title for the y axis. **ylim** a numeric vectors of length 2, giving the y coordinate range.
`q`	an integer (optional), the index of the functional covariate to plot.

Examples

```
library(RColorBrewer)
data(data1)
data(param1)
data(res_bliss1)
param1$cols <- colorRampPalette(brewer.pal(9,"Reds"))(1e2)
image_Bliss(res_bliss1$beta_posterior_density,param1,q=1)
lines(res_bliss1$data$grids[[1]],res_bliss1$Bliss_estimate[[1]],type="s",lwd=2)
lines(res_bliss1$data$grids[[1]],res_bliss1$data$betas[[1]],col=3,lwd=2,type="s")

# ---- not run
param1$cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(1e2)
image_Bliss(res_bliss1$beta_posterior_density,param1,q=1)
lines(res_bliss1$data$grids[[1]],res_bliss1$Bliss_estimate[[1]],type="s",lwd=2)
lines(res_bliss1$data$grids[[1]],res_bliss1$data$betas[[1]],col=3,lwd=2,type="s")

param1$cols <- rev(heat.colors(12))
param1$col_scale <- "quantile"
image_Bliss(res_bliss1$beta_posterior_density,param1,q=1)
lines(res_bliss1$data$grids[[1]],res_bliss1$Bliss_estimate[[1]],type="s",lwd=2)
lines(res_bliss1$data$grids[[1]],res_bliss1$data$betas[[1]],col=3,lwd=2,type="s")

param1$cols <- rev(terrain.colors(12))
image_Bliss(res_bliss1$beta_posterior_density,param1,q=1)
lines(res_bliss1$data$grids[[1]],res_bliss1$Bliss_estimate[[1]],type="s",lwd=2)
lines(res_bliss1$data$grids[[1]],res_bliss1$data$betas[[1]],col=2,lwd=2,type="s")

param1$cols <- rev(topo.colors(12))
image_Bliss(res_bliss1$beta_posterior_density,param1,q=1)
lines(res_bliss1$data$grids[[1]],res_bliss1$Bliss_estimate[[1]],type="s",lwd=2)
lines(res_bliss1$data$grids[[1]],res_bliss1$data$betas[[1]],col=2,lwd=2,type="s")
```

integrate_trapeze *integrate_trapeze*

Description

Trapezoidal rule to approximate an integral.

Usage

```
integrate_trapeze(x, y)
```

Arguments

x	a numerical vector, the discretization of the domain.
y	a numerical value, the discretization of the function to integrate.

Value

a numerical value, the approximation.

Examples

```
x <- seq(0,1,le=1e2)
integrate_trapeze(x,x^2)

integrate_trapeze(data1$grids[[1]],t(data1$x[[1]]))
```

`interpretation_plot` *interpretation_plot*

Description

Provide a graphical representation of the functional data with a focus on the detected periods with the Bliss method.

Usage

```
interpretation_plot(data, Bliss_estimate, q = 1, centered = FALSE, cols = NULL)
```

Arguments

`data`	a list containing: **y** a numerical vector, the outcomes. **x** a list of matrices, the qth matrix contains the observations of the qth functional covariate at time points given by `grids`. **grids** a list of numerical vectors, the qth vector is the grid of time points for the qth functional covariate.
`Bliss_estimate`	a numerical vector, the Bliss estimate.
`q`	an integer (optional), the index of the functional covariate to plot.
`centered`	a logical value (optional), If TRUE, the functional data are centered.
`cols`	a numerical vector of colours (optional).

Examples

```
data(data1)
data(param1)
# result of res_bliss1 <- fit_Bliss(data=data1,param=param1,verbose=TRUE)
data(res_bliss1)
interpretation_plot(data=data1,Bliss_estimate=res_bliss1$Bliss_estimate,q=1)
interpretation_plot(data=data1,Bliss_estimate=res_bliss1$Bliss_estimate,q=1,centered=TRUE)
```

`lines_bliss` *lines_bliss*

Description

A suitable representation of the Bliss estimate.

Usage

```
lines_bliss(x, y, connect = FALSE, ...)
```

Arguments

x	the coordinates of points in the plot.
y	the y coordinates of points in the plot.
connect	a logical value (optional), to handle discontinuous function. If connect is TRUE, the plot is one line. Otherwise, several lines are used.
...	Arguments to be passed to methods, such as graphical parameters (see par).

Examples

```
### Plot the BLiss estimate on a suitable grid

data(data1)
data(param1)
# res_bliss1 <- fit_Bliss(data=data1,param=param1,verbose=TRUE)

data(res_bliss1)
### Plot the BLiss estimate on a suitable grid
plot_bliss(res_bliss1$data$grids[[1]],
           res_bliss1$Bliss_estimate[[1]],lwd=2,bound=FALSE)
lines_bliss(res_bliss1$data$grids[[1]],
            res_bliss1$Smooth_estimate[[1]],lty=2)
```

---

param1 *A list of param for bliss model*

Description

A list of param for bliss model

Usage

```
param1
```

Format

a list of param for bliss model

Q the number of functional covariates

n the sample size

p the number of observation times

beta_shapes the shapes of the coefficient functions

grids_lim the range of the observation times

grids the grids of the observation times

K the number of intervals for the coefficient function

pdexp *pdexp*

Description

Probability function of a discretized Exponentiel distribution.

Usage

```
pdexp(a, l_values)
```

Arguments

a	a positive value, the mean of the Exponential prior.
l_values	a numerical value, the discrete support of the parameter l.

Value

a numerical vector, which is the prability function on `l_values`.

Examples

```
pdexp(10,seq(0,1,1))

x <- seq(0,10,le=1e3)
plot(x,dexp(x,0.5),lty=2,type="l")
lines(pdexp(0.5,1:10),type="p")
```

plot_bliss *plot_bliss*

Description

A suitable representation of the Bliss estimate.

Usage

```
plot_bliss(x, y, connect = FALSE, xlab = "", ylab = "", ylim = NULL, ...)
```

Arguments

x	the coordinates of points in the plot.
y	the y coordinates of points in the plot.
connect	a logical value (optional), to handle discontinuous function. If connect is TRUE, the plot is one line. Otherwise, several lines are used.
xlab	a title for the x axis.
ylab	a title for the y axis.
ylim	a numeric vectors of length 2, giving the y coordinate range.
...	Arguments to be passed to methods, such as graphical parameters (see par).

Examples

```
data(data1)
data(param1)
# res_bliss1 <- fit_Bliss(data=data1,param=param1,verbose=TRUE)

data(res_bliss1)
### Plot the BLiss estimate on a suitable grid
plot_bliss(res_bliss1$data$grids[[1]],
           res_bliss1$Bliss_estimate[[1]],lwd=2,bound=FALSE)
```

printbliss *Print a bliss Object*

Description

Print a bliss Object

Usage

```
printbliss(x, ...)
```

Arguments

x	input bliss Object
...	further arguments passed to or from other methods

Examples

```
# See fit_Bliss() function
```

`prior_Bliss` *fit_Bliss*

Description

Fit the Bayesian Functional Linear Regression model (with Q functional covariates).

Usage

```
prior_Bliss(data, param, support_estimate = TRUE, verbose = FALSE)
```

Arguments

`data` a list containing:

- **Q** an integer, the number of functional covariates.
- **y** a numerical vector, the outcomes.
- **x** a list of matrices, the qth matrix contains the observations of the qth functional covariate at time points given by `grids`.
- **grids** a list of numerical vectors, the qth vector is the grid of time points for the qth functional covariate.

`param` a list containing:

- **iter** an integer, the number of iterations of the Gibbs sampler algorithm.
- **K** a vector of integers, corresponding to the numbers of intervals for each covariate.
- **basis** a character vector (optional). The possible values are "uniform" (default), "epanechnikov", "gauss" and "triangular" which correspond to different basis functions to expand the coefficient function and the functional covariates
- **burnin** an integer (optional), the number of iteration to drop from the posterior sample.
- **iter_sann** an integer (optional), the number of iteration of the Simulated Annealing algorithm.
- **k_max** an integer (optional), the maximal number of intervals for the Simulated Annealing algorithm.
- **l_max** an integer (optional), the maximal interval length for the Simulated Annealing algorithm.
- **lims_kde** an integer (optional), correspond to the `lims` option of the `kde2d` funtion.
- **n_chains** an integer (optional) which corresponds to the number of Gibbs sampler runs.
- **new_grids** a list of Q vectors (optional) to compute beta samples on different grids.
- **Temp_init** a nonnegative value (optional), the initial temperature for the cooling function of the Simulated Annealing algorithm.
- **thin** an integer (optional) to thin the posterior sample.

times_sann an integer (optional), the number of times the algorithm will be executed

`support_estimate`
a logical value. If TRUE, the estimate of the coefficient function support is computed. (optional)

`verbose` write stuff if TRUE (optional).

Value

return a list containing:

alpha a list of Q numerical vector. Each vector is the function alpha(t) associated to a functional covariate. For each t, alpha(t) is the posterior probabilities of the event "the support covers t".

beta_posterior_density a list of Q items. Each item contains a list containing information to plot the posterior density of the coefficient function with the `image` function.

- `grid_t` a numerical vector: the x-axis.
- `grid_beta_t` a numerical vector: the y-axis.
- `density` a matrix: the z values.
- `new_beta_sample` a matrix: beta sample used to compute the posterior densities.

beta_sample a list of Q matrices. The qth matrix is a posterior sample of the qth functional covariates.

Bliss_estimate a list of numerical vectors corresponding to the Bliss estimates of each functional covariates.

chains a list of `posterior_sample`. `chains` is `NULL` if `n_chains=1`.

chains_info a list for each chain providing: a mu estimate, a sigma_sq estimate, the Smooth estimate of the coefficient function and the autocorrelation of the Markov Chain.

data a list containing the data.

posterior_sample a list of information about the posterior sample: the trace matrix of the Gibbs sampler, a list of Gibbs sampler parameters and the posterior densities.

support_estimate a list of support estimates of each functional covariate.

support_estimate_fct another version of the support estimates.

trace_sann a list of Q matrices which are the trace of the Simulated Annealing algorithm.

Examples

```
# see the vignette BlissIntro.
```

`res_bliss1` *A result of the BliSS method*

Description

A result of the BliSS method

Usage

```
res_bliss1
```

Format

a Bliss object (list)

alpha a list of Q numerical vector. Each vector is the function alpha(t) associated to a functional covariate. For each t, alpha(t) is the posterior probabilities of the event "the support covers t".

beta_posterior_density a list of Q items. Each item contains a list containing information to plot the posterior density of the coefficient function with the `image` function.

- `grid_t` a numerical vector: the x-axis.
- `grid_beta_t` a numerical vector: the y-axis.
- `density` a matrix: the z values.
- `new_beta_sample` a matrix: beta sample used to compute the posterior densities.

beta_sample a list of Q matrices. The qth matrix is a posterior sample of the qth functional covariates.

Bliss_estimate a list of numerical vectors corresponding to the Bliss estimates of each functional covariates.

chains_info a list containing (for each chain): a mu estimate, a sigma_sq estimate, the Smooth estimate of the coefficient function and the autocorrelation of the Markov Chain.

data see the description of the object `data1`.

posterior_sample a list containing (for each chain) the result of the `Bliss_Gibbs_Sampler` function.

Smooth_estimate a list containing the Smooth estimates of the coefficient functions.

support_estimate a list containing the estimations of the support.

support_estimate_fct a list containing the estimation of the support.

trace_sann a list containing (for each chain) the trace of the Simulated Annealing algorithm.

sigmoid *sigmoid*

Description

Compute a sigmoid function.

Usage

```
sigmoid(x, asym = 1, v = 1)
```

Arguments

x	a numerical vector, time points.
asym	a numerical value (optional), the asymptote of the sigmoid function.
v	a numerical value (optional), related to the slope at the origin.

Details

see the function `sim_x`.

Value

a numerical vector.

Examples

```
## Test 1 :
x <- seq(-7,7,0.1)
y <- sigmoid(x)
plot(x,y,type="l",main="Sigmoid function")
## Test 2 :
x  <- seq(-7,7,0.1)
y  <- sigmoid(x)
y2 <- sigmoid(x,asym=0.5)
y3 <- sigmoid(x,v   =  5)
plot(x,y,type="l",main="Other sigmoid functions")
lines(x,y2,col=2)
lines(x,y3,col=3)
```

`sigmoid_sharp` *sigmoid_sharp*

Description

Compute a sharp sigmoid function.

Usage

```
sigmoid_sharp(x, loc = 0, ...)
```

Arguments

`x`	a numerical vector, time points.
`loc`	a numerical value (optional), the time of the sharp.
`...`	Arguments (optional) for the function sigmoid.

Details

see the function `sim_x`.

Value

a numerical vector.

Examples

```
## Test 1 :
x <- seq(-7,7,0.1)
y <- sigmoid_sharp(x)
plot(x,y,type="l",main="Sharp sigmoid")
## Test 2 :
x  <- seq(-7,7,0.1)
y  <- sigmoid_sharp(x,loc=3)
y2 <- sigmoid_sharp(x,loc=3,asym=0.5)
y3 <- sigmoid_sharp(x,loc=3,v   =  5)
plot(x,y,type="l",main="Other sharp sigmoids")
lines(x,y2,col=2)
lines(x,y3,col=3)
```

`sim` *sim*

Description

Simulate a dataset for the Function Linear Regression model.

Usage

```
sim(param, verbose = FALSE)
```

Arguments

`param` a list containing:

- **beta_shapes** a character vector. The qth item indicates the shape of the coefficient function associated to the qth functional covariate.
- **n** an integer, the sample size.
- **p** a vector of integers, the qth component is the number of times for the qth covariate.
- **Q** an integer, the number of functional covariates.
- **autocorr_diag** a list of numerical vectors (optional), the qth vector is the diagonal of the autocorrelation matrix of the qth functional covariate.
- **autocorr_spread** a vector of numerical values (optional) which are related to the autocorrelation of the functional covariates.
- **grids** a list of numerical vectors (optional), the qth vector is the grid of time points for the qth functional covariate.
- **grids_lim** a list of numerical vectors (optional), the qth item is the lower and upper boundaries of the domain for the qth functional covariate.
- **link** a function (optional) to simulate data from the Generalized Functional Linear Regression model.
- **mu** a numerical value (optional), the 'true' intercept of the model.
- **r** a nonnegative value (optional), the signal to noise ratio.
- **x_shapes** a character vector (optional). The qth item indicates the shape of the functional covariate observations.

`verbose` write stuff if TRUE.

Value

a list containing:

- **Q** an integer, the number of functional covariates.
- **y** a numerical vector, the outcome observations.
- **x** a list of matrices, the qth matrix contains the observations of the qth functional covariate at time points given by `grids`.

- **grids** a list of numerical vectors, the qth vector is the grid of time points for the qth functional covariate.
- **betas** a list of numerical vectors, the qth vector is the 'true' coefficient function associated to the qth covariate on a grid of time points given with `grids`.

Examples

```
library(RColorBrewer)
param <- list(Q=2,n=25,p=c(50,50),grids_lim=list(c(0,1),c(-1,2)))
data <- sim(param)
data$y
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(10)
q=2
matplot(data$grids[[q]],t(data$x[[q]]),type="l",lty=1,col=cols)
plot(data$grids[[q]],data$betas[[q]],type="l")
abline(h=0,lty=2,col="gray")
```

`sim_x` *sim_x*

Description

Simulate functional covariate observations.

Usage

```
sim_x(param)
```

Arguments

`param` a list containing :

- **grid** a numerical vector, the observation times.
- **n** an integer, the sample size.
- **p** an integer, the number of observation times.
- **diagVar** a numerical vector (optional), the diagonal of the autocorrelation matrix.
- **dim** a numerical value (optional), the dimension of the Fourier basis, if "shape" is "Fourier" or "Fourier2".
- **ksi** a numerical value (optional) related to the observations correlation.
- **x_shape** a character vector (optional), the shape of the observations.

Details

Several shape are available for the observations: "Fourier", "Fourier2", "random_walk", "random_sharp", "uniform", "gaussian", "mvgauss", "mvgauss_different_scale", "mvgauss_different_scale2", "mvgauss_different_scale3" and "mvgauss_different_scale4".

Value

a matrix which contains the functional covariate observations at time points given by `grid`.

Examples

```
library(RColorBrewer)
### Fourier
param <- list(n=15,p=100,grid=seq(0,1,length=100),x_shape="Fourier")
x <- sim_x(param)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(15)
matplot(param$grid,t(x),type="l",lty=1,col=cols)
### Fourier2
param <- list(n=15,p=100,grid=seq(0,1,length=100),x_type="Fourier2")
x <- sim_x(param)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(15)
matplot(param$grid,t(x),type="l",lty=1,col=cols)
### random_walk
param <- list(n=15,p=100,grid=seq(0,1,length=100),x_type="random_walk")
x <- sim_x(param)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(15)
matplot(param$grid,t(x),type="l",lty=1,col=cols)
### random_sharp
param <- list(n=15,p=100,grid=seq(0,1,length=100),x_type="random_sharp")
x <- sim_x(param)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(15)
matplot(param$grid,t(x),type="l",lty=1,col=cols)
### uniform
param <- list(n=15,p=100,grid=seq(0,1,length=100),x_type="uniform")
x <- sim_x(param)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(15)
matplot(param$grid,t(x),type="l",lty=1,col=cols)
### gaussian
param <- list(n=15,p=100,grid=seq(0,1,length=100),x_type="gaussian")
x <- sim_x(param)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(15)
matplot(param$grid,t(x),type="l",lty=1,col=cols)
### mvgauss
param <- list(n=15,p=100,grid=seq(0,1,length=100),x_type="mvgauss")
x <- sim_x(param)
cols <- colorRampPalette(brewer.pal(9,"YlOrRd"))(15)
matplot(param$grid,t(x),type="l",lty=1,col=cols)
```

`support_estimation` *support_estimation*

Description

Compute the support estimate.

Usage

```
support_estimation(beta_sample_q, gamma = 0.5)
```

Arguments

`beta_sample_q`	a matrix. Each row is a coefficient function computed from the posterior sample.
`gamma`	a numeric value, the default value is `0.5`.

Value

a list containing:

alpha a numerical vector. The approximated posterior probabilities that the coefficient function support covers `t` for each time points `t`.

estimate a numerical vector, the support estimate.

estimate_fct a numerical vector, another version of the support estimate.

Examples

```
data(data1)
data(param1)
# result of res_bliss1<-fit_Bliss(data=data1,param=param1)
data(res_bliss1)
res_support <- support_estimation(res_bliss1$beta_sample[[1]])

### The estimate
res_support$estimate
### Plot the result
grid <- res_bliss1$data$grids[[1]]
plot(grid,res_support$alpha,ylim=c(0,1),type="l",xlab="",ylab="")
for(k in 1:nrow(res_support$estimate)){
    segments(grid[res_support$estimate[k,1]],0.5,
             grid[res_support$estimate[k,2]],0.5,lwd=2,col=2)
    points(grid[res_support$estimate[k,1]],0.5,pch="|",lwd=2,col=2)
    points(grid[res_support$estimate[k,2]],0.5,pch="|",lwd=2,col=2)
}
abline(h=0.5,col=2,lty=2)
```

`%between%` *between*

Description

Check if a number belong to a given interval.

Usage

```
value %between% interval
```

Arguments

`value`	a numerical value.
`interval`	a numerical vector: (lower,upper).

Value

a logical value.

Examples

```
1 %between% c(0,2)
2 %between% c(0,2)
3 %between% c(0,2)
```

Index